《编译技术》

课程设计文档

学号：\_\_\_\_14231027\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_赵岳\_\_\_\_\_\_\_\_\_\_

2016年11 月11 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＞｝  
＜整数＞        ::= ［＋｜－］＜无符号整数＞｜０  
＜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＞‘]’ )}  
＜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＞‘)’ ‘{’＜复合语句＞‘}’  
＜无返回值函数定义＞  ::= void＜标识符＞‘(’＜参数＞‘)’‘{’＜复合语句＞‘}’  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
＜参数＞    ::= ＜参数表＞  
＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞  
＜主函数＞    ::= void main‘(’‘)’‘{’＜复合语句＞‘}’  
＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜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类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符区分大小写字母

（3）写语句中的字符串原样输出

（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

***（6）本文法无for循环，但编译器内仍将for保留为关键字，不能声明以“for”为名字的标识符。***

### 2．目标代码说明

生成MIPS汇编码。所有指令和功能与MIPS指令集保持一致。

### 3. 优化方案\*

【说明需要完成的优化方案及其要求】

## 二．详细设计

### 1．程序结构

### 绋嬪簭缁撴瀯

* 出错处理在词法分析和语法分析中均有。例如，在词法分析中进行一些以0开头的正整数的报错、非法常量字符、字符串超长等错误；在语法分析中则会抛出关于一些语法结构方面的错误。
* 符号表管理与语法分析一并进行，不进行额外的符号表管理环节。
* 中间代码选择四元式。
* 目标代码选择Mips汇编。

### 2．类/方法/函数功能

# 词法分析类Lexer（实例化为lexer）

在此类的构造函数内初始化保留字表。保留字表采用c++ map的功能实现。

此类的方法有：

void getch(); //源程序中读取字符。

void getsym(); //获取一个单词，并将该单词归类或报错

# 语法分析类Parser（实例化为parser）

此类的构造函数内初始化各个分程序的first集合，用于出错判断和处理。

void test(set<Symbol> s1, set<Symbol> s2, int n); //assert，s1合法集，s2停止集合

void testsemicolon(set<Symbol> fsys);//检测分号，并读入分号后的字符，停在fsys

void testsemicolon\_and\_rbrace(set<Symbol> fsys);//检测分号和右大括号，同上

void testrbrace(set<Symbol> fsys);//检测右大括号

void skip(set<Symbol> fsys, int n);//跳读并报错，并停到fsys，错误号n

void program(set<Symbol> fsys);//program//程序分析

void constantdeclaration(set<Symbol> fsys);//常量声明

void constantdefine(set<Symbol> fsys);//常量定义

void var\_varfuncdeclaration(set<Symbol> fsys);//变量、函数声明

void var\_varfuncdefine(set<Symbol> fsys);//变量、函数定义

bool HaveReachfunc;//到达函数声明flag

void voidfunc\_maindeclaration(set <Symbol> fsys);//void型函数声明

void voidfunc\_maindefine(set<Symbol> fsys);//void型函数定义

bool HaveReachMain;//到达main函数flag

void compstate(set <Symbol> fsys);//复合语句

void constantdeclaration\_in\_func(set<Symbol> fsys);//常量声明

void constantdefine\_in\_func(set<Symbol> fsys);//常量定义

void vardeclaration(set<Symbol> fsys);//变量声明

void vardefine(set<Symbol> fsys);//变量定义

void statement(set<Symbol> fsys);//语句

void ifstatement(set<Symbol> fsys);//if语句

void scanfstatement(set<Symbol> fsys);//scanf语句

void printfstatement(set<Symbol> fsys);//printf语句

void returnstatement(set<Symbol> fsys);//return语句

void whilestatement(set<Symbol> fsys); //while语句

void assign\_or\_callstatement(set<Symbol> fsys); //赋值或调用函数语句

void valueparalist(set<Symbol> fsys); //值参数表

void switchstatement(set<Symbol> fsys); //情况语句

void casestatement(set<Symbol> fsys); //case分支语句

void defaultstatement(set<Symbol> fsys);//缺省语句

………..未完待续

该类方法过多未列举完。Parser类中的函数较为简单，主要是完成递归下降的语法分析过程。

# 错误处理类Error（实例化为error）

void error(int n);//错误信息输出

void test();//错误处理函数

# 符号表管理类Table（实例化为table）

void enter(…); //待设计

void enterFuncinfo(int current, vector<Pitem> plist);//加入funtion的info

void enterArray(int current, int index, Aitem aitem);//加入数组

int enterString(string s);//将conststring enter进表

int newTable(int current); //新建一个表

int delTable(int current); //删除一个表

int index(int current);//返回ident位置

bool exist(int current, string ident);//查询是否在表

Item\* complete(int current, int index);

Item get(int current, string ident, int& tabx, int& index);

PFinfo getPFinfo(Item item);

Aitem getArray(Item item);

Item find(int tabx, int index);

string findString(int index);

int getStabSize();

# 四元式类Quarter、Operand（实例化为quater）

enum Operation { //四元式操作（具体见后文四元式设计）

ADD, SUB, MUL, DIV, NEG,

BEQ, BNE, BGT, BGE, BLT, BLE,

JMP, OFF, SEL, ASG, RAD, WRT,

CALP, CALF, RET, VAL, PUSH, SAVE, LAB

};

enum Opdtype { //操作数类型

oconst, oident, otemp, olabel, ostring

};

Operand::Operand(); //构造函数

Operand::string getName(); //查符号表得到标识符名字或

Operand::string toString(); //将操作符的其余部分转化为字符串

Quarter::Quater(); //构造函数

Quarter::Quater(/\*fix para here\*/);//完成四元式后面的部分

string toString(); //转化为字符串的函数

void outputIns(); //将四元式转储在文件中

### 3．调用依赖关系

首先，上述的类会在global.cpp中实例化，在main函数中，会调用Parser::parse方法。该方法是编译程序的入口。具体的语法分析树状图可以大致如下表示：（某些细节函数需要在最终实现时确定）

在statement后，由于语法的first集合可以很好的分离，因此按照文法，进行递归下降子程序的语法分析即可。

在语法分析的过程中，还需要进行错误处理和符号表、函数表的填充，以及四元式的生成。完成的步骤主要如下：

* 先完成语法分析，保证能够正确地将源代码parse成我们需要的形式，也即根据文法进入到正确的递归下降子程序中，此步直接完成关于语法的错误处理。
* 再完成上一步的基础上，对有变量、函数声明的递归下降子程序中填充相应的填表工作。此时需要Table类。除此以外，此步还会产生语义错误处理。比如，函数的返回值类型错误、不合法的调用、类型和赋值的不匹配等。
* 再上述工作都完成后，便可以进行四元式的生成，此时需要Quarter类。

### 4．符号表管理方案

符号表用来管理标识符的名字和类型，并且进行检查。符号表为栈式符号表，采用vector实现，源程序每一层次生成一系列符号表项，同时将每一层符号表项第一项的索引存在一个栈中，实现栈式管理。

符号表的表项也设计为一个类，其每一个表项应具有的属性有：

1. Name（名字）
2. Kind（种类）

const、var、pro、fun、array、para

1. Type（类型——仅对fun、var、array、para有效）

char、integer

1. Level（层次）
2. Addr（首地址）
3. OtherInfo（附加信息）

Const——value

Array——length

Function——arg list

### 5．存储分配方案

采用动态存储分配方案，因为根据文法的定义允许程序、函数的嵌套调用，所以需要采用动态存储分配和管理的方案，具体来讲就是为每一层次分配一个活动记录。如上图所示。

在这里要定义的运行栈其实是活动记录的栈，其数据结构采用C++中的vector类，vector中的元素为活动记录对应的类。

具体的存储分配方案由Mars决定。根据Mips的相关文档，详细如下：

# 5.1调用之前

返回地址保到$ra

将$s0-$7存入当前活动记录

存入传入参数（直接保存到所调用的函数的活动记录）

# 5.2调用开始

$fp存入运行栈中的Pre $fp

$ra存入ret address

$sp给$fp

$sp -= space（栈向下增长）

# 5.3函数返回

返回值存入$v0,$v1

$s0-$s7全局变量保存

$ra = ret address

$sp=$fp;

$fp=pre $fp

Jr

# 5.4函数调用后

$s0-$s7恢复

### 6. 解释执行程序\*

采用Mars模拟器。

### 7. 四元式设计\*

运算操作符：

ADD s1,s2,d d := s1 + s2

SUB s1,s2,d d := s1 - s2

MUL s1,s2,d d := s1 \* s2

DIV s1,s2,d d := s1 / s2

NEG s1,d d := -s1

比较操作符：

LSS s1,s2,d d := s1 < s2

LEQ s1,s2,d d := s1 <= s2

GTR s1.s2.d d := s1 > s2

GEQ s1,s2,d d := s1 >= s2

EQL s1,s2,d d := s1 = s2

NEQ s1.s2.d d := s1 <>s2

读写操作符：

RAD d 将内容读至d

WRT d 将d的内容输出

跳转操作符：

JMP d 无条件跳转至指令地址d处

JPC s1,d 当s1为假时跳转至指令地址d处

CALF s1,d 调用s1有返回值函数，d为返回值

CALFV s1 调用s1有返回值函数，d为返回值

RET a1 return语句的四元式，a1若为空即为void函数

其他：

RETM main函数返回语句。

### 8. 目标代码生成方案\*

void Parser::outputAsm();

该函数主要用于Mips代码的生成。将该函数的主要作用做如下说明：

1. 首先声明stringstream ss;用于存储每一条mips指令。
2. 完成准备工作。
3. 只要四元式文件或vector中还有内容，就不断循环。
4. 根据四元式的操作符不同，生成不同的Mips代码。

### 9. 优化方案\*

【说明代码优化有关的数据结构、关键算法】

### 10. 出错处理

/\* 0\*/ "源文件开始有非法单词",

/\* 1\*/ "由于代码不全，以读至代码文件结束",

/\* 2\*/ "无符号整数不能以0开始",

/\* 3\*/ "+\\-号后丢失数值",

/\* 4\*/ "不合法的字符常量",

/\* 5\*/ "缺少右大括号",

/\* 6\*/ "字符常量为空",

/\* 7\*/ "缺少左大括号",

/\* 8\*/ "const的声明需注明常量类型（已跳转至下一条合法常量声明）",

/\* 9\*/ "缺少右大括号或分号",

/\*10\*/ "缺少分号",

/\*11\*/ "缺少逗号",

/\*12\*/ "无法识别标识符是变量、数组还是带返回值函数",

/\*13\*/ "缺少冒号",

/\*14\*/ "函数的开始有非法单词",

/\*15\*/ "缺少左中括号",

/\*16\*/ "应该是一个数字",

/\*17\*/ "缺少右中括号",

/\*18\*/ "无法识别标识符是变量还是数组",

/\*19\*/ "语句内不能进行常量、变量的定义",

/\*20\*/ "应为标识符",

/\*21\*/ "数组上界应为正整数",

/\*22\*/ "无效字符",

/\*23\*/ "缺少右小括号",

/\*24\*/ "不等号书写错误（编译时已更正）",

/\*25\*/ "void 关键词后应为标识符或main",//

/\*26\*/ "语句的开始有非法单词",

/\*27\*/ "常量非法赋值",//

/\*28\*/ "应该是while",

/\*29\*/ "缺少赋值符号",

/\*30\*/ "常量只进行了声明没有赋值（缺少等号）",//

/\*31\*/ "常量、变量和数组的声明应在函数声明之前（但仍视作该次声明有效）",//

/\*32\*/ "缺少左小括号",

/\*33\*/ "缺少双引号",

/\*34\*/ "本编译器不支持for循环",

/\*35\*/ "printf语句只能输出表达式和字符串",

/\*36\*/ "非法的表达式",

/\*37\*/ "非法的项",

/\*38\*/ "void函数没有函数名",//

/\*39\*/ "不能对标识符重新定义",

/\*40\*/ "未定义的标识符",

/\*41\*/ "缺少case关键字",

/\*42\*/ "case后只能使用常量",

/\*43\*/ "缺少冒号",

/\*44\*/ "类型标识符只能是int或char",

/\*45\*/ "无法识别该语句为赋值还是函数调用",

/\*46\*/ "缺少单引号",

/\*47\*/ "重复定义标识符", // 语义错误

/\*48\*/ "未定义的标识符",

/\*49\*/ "数组索引类型应为INT",

/\*50\*/ "赋值类型不匹配",

/\*51\*/ "因子内的函数调用必须有返回值",

/\*52\*/"void函数不应有返回值",

/\*53\*/"scanf变量不能为函数"

说明：

出错处理本着局部化的原则。首先，错误可以准确定位到源代码的某行。其次，遇到错误会跳出到某个子程序正常退出的符号上，保证后面语法分析的顺利进行，实现错误局部化。

具体实现的相关函数：

void Parser::test(set<Symbol> s1, set<Symbol>s2, int n) {

if (s1.find(sym) != s1.end()) {

return;

}

error.outputError(n);

s1.insert(s2.begin(), s2.end());

while (s1.find(sym) == s1.end()) { //until sym in s1 or s2

lexer.getsym();

}

}

void Parser::testsemicolon(set<Symbol> fsys) { //REMEMBER : This func will read the next sym after ;

if (sym == SEMICOLON) {

lexer.getsym();

return;

}

else {

skip(fsys,10);

}

}

void Parser::testsemicolon\_and\_rbrace(set<Symbol> fsys) { //REMEMBER : This func will read the next sym after ; and }

if (sym == SEMICOLON||sym==RBRACE) {

lexer.getsym();

return;

}

else {

skip(fsys,9);

}

}

void Parser::testrbrace(set<Symbol> fsys) { //REMEMBER : This func will read the next sym after ;

if (sym == RBRACE) {

lexer.getsym();

return;

}

else {

skip(fsys,5);

}

}

void Parser::skip(set<Symbol> fsys, int n) {

error.outputError(n);

while (fsys.find(sym) == fsys.end()) { //never find such a sym in fsys stop at first meet fsys

lexer.getsym();

}

}

每个函数都有fsys参数，作为错误处理的停止符号集。由于分号、大括号是经常需要检测的关键字符，所以抽象为函数。其余的字符检测放在了其他函数内。

## 三．操作说明

### 1．运行环境

Window XP sp3及以上操作系统

安装Visual Studio2010。

安装JAVA运行环境 JRE。

安装Mips运行模拟器Mars。

### 2．操作步骤

运行Release版本的编译器C0compiler.exe。输入带有路径的待编译文件名并回车。

如果控制台输出：

Compile completed!

Output table completed!

Output quater completed!

Output ASM code completed!

则说明编译并生成目标代码成功。目标代码存储在相对路径的Asm.asm文件中。

如果控制台输出：

Compile failed!

Find 2 errors in total

第 120行 scanf变量不能为函数

第 121行 void函数不应有返回值

则说明编译失败，原因是源程序有错误。错误信息将显示在控制台上。

## 四．测试报告

### 1．测试程序及测试结果

#### C1:

const char c='a';

void main(){

int i;

char b;

scanf(i);

scanf(b);

printf("Hello world!");

printf(c+1);

return;

}

输入：1 a

输出：Hello world!b

#### C2:

int gcd(int a , int b){

if (a == b) {

return (a);

}

if (a > b) {

return (gcd(b, a - b));

}

if (a < b) {

return (gcd(b - a, a));

}

}

void main(){

int a,b;

scanf(a,b);

printf("GCD a,b is:",gcd(a,b));

return;

}

输入：

10 2

输出：

GCD a,b is:2

输入：

50 15

输出：

GCD a,b is:5

#### C3:

int glbA;

int fib(int x){

if(x==1)

return (1);

if(x==2)

return (1);

return (fib(x-1)+fib(x-2));

}

void main(){

int x;

int arr[100];

printf("Please input a num:\n");

scanf(x);

printf("Fib(x) num is ", fib(x));

x=20;

arr[x+1]=100;

return;

}

输出：

Please input a num:

6

Fib(x) num is 8

输出：

Please input a num:

1

Fib(x) num is 1

#### C4：

const int NUMBER=2;

const char cha = '\*';

int targetvar;

int ADD(int a,int b){

return (+a+b);

}

int fib(int x){

if(x<=0){

printf("fib para error! return ERROR CODE = -1\t");

return (-1);

}

if(x==1)

return (1);

if(x==2)

return (1);

return ((fib(x-1)+fib(x-2)));

}

void main(){

const int TEN = 10;

int option,i;

int x,y;

int chaint;

int array[10];

char cha\_;

i=1;

printf("\n",ADD(x,y));

printf("\n");

while(i<TEN){

chaint=cha+i;

cha\_=cha+i;

printf("\t", cha\_);

printf("\t",chaint);

printf("\t\t");

i=i+1;

}

printf("\nFib sum is :",fib(ADD(x,y)+1)+10);

printf("\nFib sum is :",fib(-ADD(x,y+x)

+1)

+1);

array[3\*6-10]=ADD(x,y+1);

array[0] = array[8]+1;

printf("\nArray[0]:",array[10\*9\*8\*0]);

return;

}

输入：

无

输出：

0

+ 43 , 44 - 45 . 46 / 47 0 48 1 49 2 50 3 51

Fib sum is :11

Fib sum is :2

Array[0]:2

#### C5:

const int NUMBER=2;

const char cha = '\*';

int targetvar;

int ADD(int a,int b){

return (+a+b);

}

void guess(int a,int b){

int guessnum;

guessnum=(a+b)/2;

if(guessnum>targetvar){

printf("\nToo large!Upper bound now is:",guessnum);

guess(a,guessnum);

return;

}

if(guessnum<targetvar){

printf("\nToo small!Lower bound now is:",guessnum);

guess(guessnum,b);

return;

}

if(guessnum==targetvar){

printf("\nCongrats!Correct number:",guessnum);

return;

}

}

void main(){

const int TEN = 10;

int option,i;

int x,y;

int chaint;

int array[10];

char cha\_;

i=1;

if(i<NUMBER){

scanf(option,x,y);

switch(option){

case 1:{targetvar=10;guess(0,20);}

case 2:{targetvar=8;guess(0,20);}

case 3:{targetvar=11;guess(0,20);}

case 4:{targetvar=16;guess(0,20);}

default:printf("\nWrong option!");

}

}

printf("\n",ADD(x,y));

printf("\n");

while(i<TEN){

chaint=cha+i;

cha\_=cha+i;

printf("\t", cha\_);

printf("\t",chaint);

printf("\t\t");

i=i+1;

}

array[3\*6-10]=ADD(x,y+1);

array[0] = array[8]+1;

printf("\nArray[0]:",array[10\*9\*8\*0]);

return;

}

输入：

3

1

2

输出：

Too small!Lower bound now is:10

Too large!Upper bound now is:15

Too large!Upper bound now is:12

Congrats!Correct number:11

3

+ 43 , 44 - 45 . 46 / 47 0 48 1 49 2 50 3 51

Array[0]:5

#### W1:

int GLA;

int f1(int a,int b){

int LCA,p;

char CH;

scanf(LCA;

CH = 't';

printf CH);

}

void main(){

f1;

return

}

编译器输出：

Compile failed!

Find 4 errors in total

第 5行 缺少右小括号

第 7行 缺少左小括号

第 10行 无法识别该语句为赋值还是函数调用

第 12行 缺少分号

#### W2:

void main(){

int x;

int y;

char c;

if(c<y){

x=x++y/2;

c=(c-1)\*/5;

while(1){

;

}

}

return;

}

编译器输出：

Compile failed!

Find 2 errors in total

第 6行 +\-号后丢失数值

第 7行 非法的项

#### W3：

const int a;

int b;

void main(){

int a;

scanf(c);

printf("eee",a);

}

编译器输出：

Compile failed!

Find 2 errors in total

第 1行 常量只进行了声明没有赋值（缺少等号）

第 5行 未定义的标识符

#### W4：

void f1(int a,char b){

return (a+b);

}

void main(){

int a;

f1(a,b);

scanf(f1);

printf("123",f1);

}

编译器输出：

Compile failed!

Find 3 errors in total

第 2行 void函数不应有返回值

第 7行 未定义的标识符

第 8行 scanf变量不能为函数

#### W5：

const int NUMBER=2

const char cha = '\*';

int targetvar;

int ADD(int a,int b){

return (+a++b);

}

void guess(int a,int b){

int guessnum;

guessnum=(a+b)/2;

if(guessnum>targetvar){

printf("\nToo large!Upper bound now is:",guessnum ;

guess a,guessnum);

return;

}

if(guessnum<targetvar){

printf("\nToo small!Lower bound now is:",guessnum);

guess(guessnum,b);

return;

}

if(guessnum==targetvar){

printf("\nCongrats!Correct number:",guessnum);

return;

}

}

int fib(x){

if x<=0){

printf("fib para error! return ERROR CODE = -1\t");

return (-1);

}

if(x==1)

return (1);

if(x==2)

return (1);

return ((fib(x-1)+fib(x-2)));

}

void main(){

const int TEN = 10;

int option,i;

int x,y;

int chaint;

int array[10];

char cha\_;

i=1;

if(i<NUMBER){

scanf(option,x,y);

switch(option){

case 1:{targetvar=10;guess(0,20);}

case 2:{targetvar=8;guess(0,20);}

case 3:{targetvar=11;guess(0,20);}

case 4:{targetvar=16;guess(0,20);}

default:printf("\nWrong option!");

}

}

printf("\n",ADD(x,y));

printf("\n");

while(i<TEN){

chaint=cha+i;

cha\_=cha+i;

printf("\t", cha\_);

printf("\t",chaint);

printf("\t\t");

i=i+1;

}

printf("\nFib sum is :",fib(ADD(x,y)+1)+10);

printf("\nFib sum is :",fib(-ADD(x,y+x)

+1)

+1);

array[3\*6-10 =ADD(x,y+1);

array[0] = array[8]+1;

printf("\nArray[0]:",array[10\*9\*8\*0]);

return;

}

编译器输出：

Compile failed!

Find 7 errors in total

第 2行 缺少分号

第 5行 +\-号后丢失数值

第 5行 缺少右小括号

第 11行 缺少右小括号

第 12行 无法识别该语句为赋值还是函数调用

第 25行 类型标识符只能是int或char

第 26行 缺少左小括号

### 2．测试结果分析

#### C1:

测试了基本的输入输出语句，正常。

#### C2:

测试了函数递归调用，正常。

#### C3:

测试了函数递归调用及条件跳转语句，正常。

#### C4:

综合测试1，正常。

#### C5:

综合测试2，正常。

综上，测试结果正确！测试涵盖了较容易出错的所有点，也涵盖了文法中所有的语句和语法。例如：条件判断、跳转、循环、switch、return等语句均有涵盖。

#### W1:

语法错误测试，主要测试在语法分析阶段出现的一些缺少相应语法项的错误。

#### W2:

语法错误测试，主要测试在较为复杂的语句中出现的语法错误。

#### W3:

语义错误测试，主要测试符号表与错误处理的配合性。

#### W4:

语义错误测试，主要测试新增的一些语义错误。

#### W5:

综合错误测试，主要测试能否实现错误跳读。

综上，编译器能够较为全面和科学地进行编译报错，并做出相应处理。

## 五．总结感想

光阴似箭，日月如梭。随着2016年的过去，一同过去的还有编译原理课程设计。此篇文档是最后一个需要提交的作业，而此篇感想又是文档的最后一个部分，此时此刻，五味杂陈。总体来说，一学期的编译器开发经历让我成长了很多。从学期最开始对编译这个词一无所知，到史老师深入浅出的讲解后略有了解，再到自己阅读相关资料和博客有了更深入的了解，再到最后自己着手写编译器，终于在此时此刻，我可以自豪地说，我已经对一般高级程序语言的编译到运行有了一个深入的理解。

说起编译器的开发过程，那可真是痛并快乐着。其实，我对C++的掌握也只是流于表面，然而由于C++非常好的面向对象特性和编译器的各个部分非常契合，所以我最终还是选择了用VS2010和C++作为开发语言。在开发过程中，C++有很多不同于C语言的规则和数据结构，合理地选择数据结构可以省去很多时间和代码。比如各种容器、集合、枚举，他们有非常好的栈的特性，无论是在错误向量集、四元式生成和ASM代码的导出时都能很方便地完成工作。在开发过程中，不仅学习到了编译原理，还学到了很多C++的知识，也提高了自己的编程能力。

再说说关于测试的东西吧。虽然每周都有测试，时间安排的很紧凑，看似每天都在为编译而添砖加瓦，实际上老师的初衷是把大的工作量分给每天（也确实是这样，到后期每天也许平均就改几行代码）。我认为这样的安排非常合理，也符合一个项目开发的节奏。老师的测试也帮我找到了一些关于错误处理的问题，以及延时槽的问题。

最后，我相信经过大量的测试和调试，我的编译器的稳定性、正确性都能达到要求的标准，也能非常好地支持所抽到的文法。