《编译技术》课程设计文 档

学号：\_\_\_\_\_15061047\_\_\_\_\_

姓名：\_\_\_\_\_\_陈舒玮\_\_\_\_\_\_

2018年 1月 13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＞‘]’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞|‘(’＜表达式＞‘)’

＜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＞［else＜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＞ ::= for‘(’＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞‘)’＜语句＞

＜步长＞ ::= ＜非零数字＞｛＜数字＞｝

＜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＞

＜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

1. for语句先执行一次循环体中的语句再进行循环变量是否越界的测试

### 2．目标代码说明

生成的目标代码为mips汇编指令。

### 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

如：

x = 1 + 99 优化为-> x = 100

x = 13 \* 5 优化为-> x = 65

1. 复制传播

如果出现形如x = y的中间代码，y可能是标识符或是数字，则在之后的中间代码之中，使用y来代替x，要注意的是如果离开当前基本块或是x、y被重新赋值，则停止代替。如果x为产生中间代码时产生的临时变量，则之后可以删除x = y这条中间代码。

1. 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。

（5）DAG图法

//dagpoint表示dag图中的一个点

struct dagpoint{

int op; //op为操作符，op值为1,2,3,4分别对应+,-,\*,/,初始为0

int no; //no节点的编号

dagpoint\* lchild; //本节点左子节点

dagpoint\*rchild; //本节点右子节点

dagpoint\*parent[]; //本节点的父节点(可能不止一个)

};

将所有的dagpoint都存在一个结构体数组中。

//point表示结点表中的一个表项

struct point{

dagpoint\* dag;//表项中的dag图节点

char \*id;//表项中的标识符

};

采用教材中的dag图导出中间代码的启发式算法进行优化。

## （6）全局寄存器分配（引用计数或着色算法）；

## （7）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

### 程序结构

getsym.h

词法分析

grammar.h

grammar.h

语法分析

建立符号表

error.h

错误处理

建立符号表

语法分析

语义分析，

生成中间代码

midcode.h

（常数合并、复制传播优化）

中间代码优化

mips.c

（窥孔优化）

目标代码优化

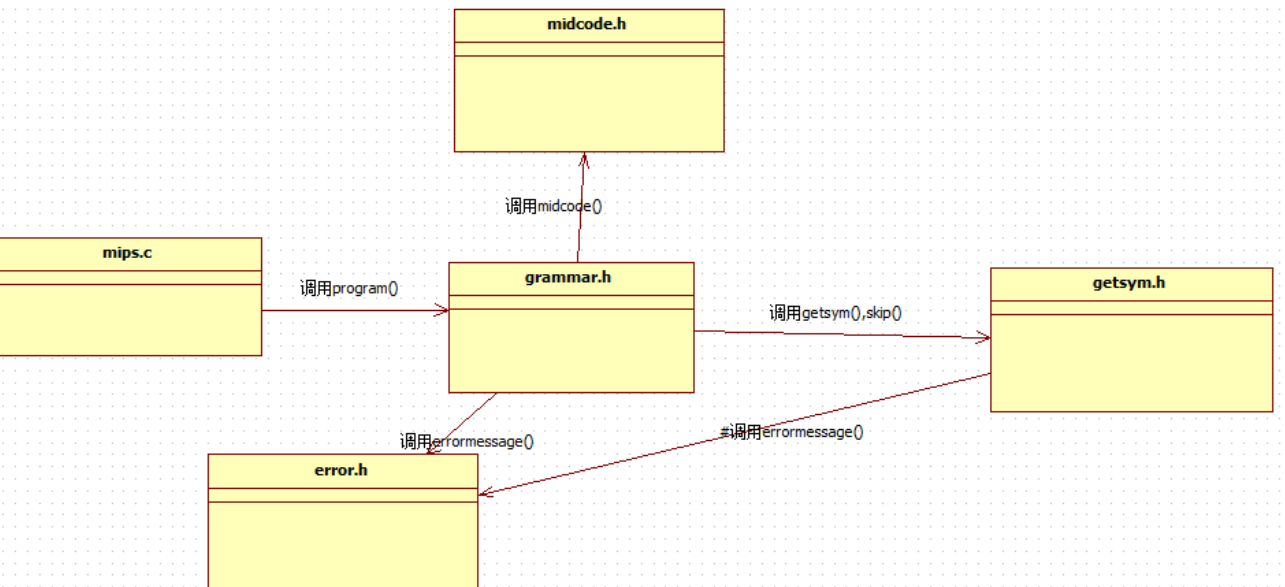
生成目标代码

### 注：grammar.c语义分析过程中，不断调用midcode.h中的函数来生成中间代码。

### 类/方法/函数功能

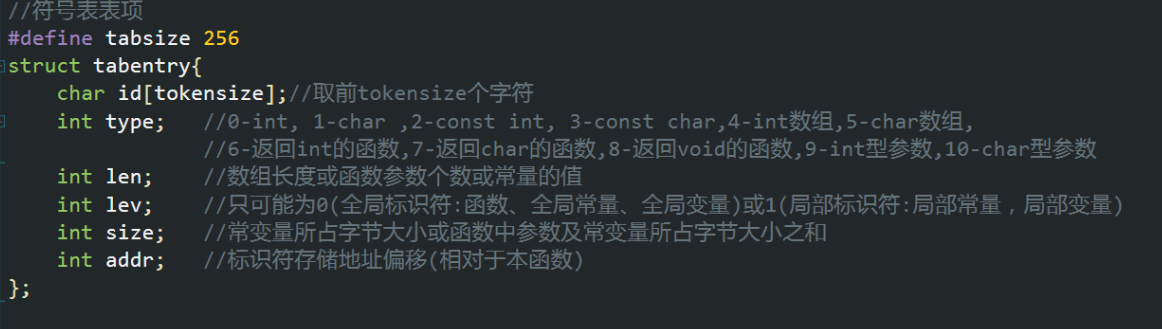
|  |  |
| --- | --- |
| getsym.h  （词法分析） | |
| void skip(int arg\_num,...) | 用于出错跳读的（变参数） |
| isreserve() | 判断当前词是否为保留字 |
| char getch() | 读取下一个字符 |
| void printtoken() | 打印当前读到的符号（调试使用） |
| void memsettoken() | 初始化token |
| getsym() | 获得下一个词 |
| grammar.h  （语法分析、语义分析） | |
| int sizecalcu(int type,int len) | 计算标识符对应类型占内存大小（返回值表示分配内存大小） |
| void inserttab(char\* id,int type,int len,int lev,int size,int addr) | 将符号插入符号表 |
| int searchtab(char\* id,int whichtype) | 语义分析时查询符号表 |
| void constdef() | 处理常量定义 |
| void conststa() | 处理常量说明 |
| void variabledef() | 处理变量定义 |
| void compoundsta() | 处理复合语句 |
| void hasretfunc() | 处理有返回值函数 |
| void noretfunc() | 处理无返回值函数 |
| void mainfunction() | 处理主函数 |
| int integer() | 处理、计算整数（返回值表示值） |
| void program() | 处理语法、语义分析 |
| int paralist() | 处理参数表（返回值表示参数个数） |
| int expression(char\* temp) | 处理表达式(返回值表示类型0-int ， 1-char) |
| void stalist() | 处理语句列 |
| void sta() | 处理语句 |
| int isstahead(int symid) | 判断是否是语句头部 |
| int term(char\* temp) | 处理项(返回值表示类型0-int ， 1-char) |
| int factor(char\* temp) | 处理因子(返回值表示类型0-int ， 1-char) |
| void valueofpara(int index) | 处理值参数表(参数为函数在符号表的索引) |
| void returnsta() | 处理返回语句 |
| void printsta() | 处理写语句 |
| void ifsta() | 处理if-else |
| void switchsta() | 处理switch-case |
| void defaultsta(char\* default\_label) | 缺省处理程序 |
| void casesubsta(char\* temp\_exp,char\* mylabel,char\* nextlabel,int switch\_type) | 处理情况表子语句 |
| void forsta() | 处理for循环 |
| void assignsta() | 处理赋值语句 |
| void callretfunc(int index) | 处理有返回值函数的调用(参数为函数在符号表的索引) |
| void callnoretfunc(int index) | 处理无返回值函数调用(参数为函数在符号表的索引) |
| int isstahead(int symid) | 语句可能的第一个符号 |
| void condition() | 处理条件 |
| midcode.h  （中间代码生成、中间代码优化） | |
| int isrealnum(char \*s) | 判断字符串是不是数字 |
| void constcombine(MIDCODE\* code) | 常数合并优化 |
| void copyconv() | 复制传播优化 |
| void gettemp(char\* str) | 生成一个临时变量 |
| void getlabel(char \* str) | 生成一个标签 |
| void midcode(int type,char\* op1,char\* op2,char\* result) | 生成中间代码 |
| void writemidcode() | 将中间代码写入midcode.txt文件 |
| error.h  （出错处理） | |
| void errormessage(int error,int line,int no) | 打印错误信息 |
| mips.c  （生成目标代码、目标代码优化） | |
| void deletelw() | 窥孔优化，删除mips代码中多余的lw语句 |
| void addnewsym(char\* id,int funcnum) | 向函数符号表中插入中间变量 |
| void printfunctab() | 打印函数符号表（调试使用） |
| void getstrname(char\* strname) | 产生字符串标号 |
| void mips\_init() | 初始化全局变量、常量、字符串，以及mips代码的.data段，计算每个函数中出现的中间变量，填入各个函数的符号表 |
| int getaddr(char\* id,int isarr,int funcnum,char\* addr) | 返回0，局部变量；返回1，全局变量 |
| void mips\_localconst(int index,int funcnum) | 处理局部常量定义（实际没有使用） |
| void mips\_add(int index,int funcnum) | 产生加法指令三个操作数一定不是数组 |
| void mips\_sub(int index,int funcnum) | 产生减法指令三个操作数一定不是数组 |
| void mips\_mul(int index,int funcnum) | 产生乘法指令三个操作数一定不是数组 |
| void mips\_div(int index,int funcnum) | 产生除法指令三个操作数一定不是数组 |
| void mips\_funcdef(int index,int funcnum) | 处理函数定义 |
| void mips\_pushpara(int index,int funcnum) | 处理函数传参 |
| void mips\_retvalue(int index,int funcnum) | 处理返回值 |
| void mips\_varassign(int index,int funcnum) | 处理变量赋值 |
| void mips\_arrassign(int index,int funcnum) | 处理为数组赋值op1[op2] = result |
| void mips\_arruse(int index,int funcnum) | 处理数组值赋给非数组变量result = op1[op2] |
| void mips\_printstr(int index) | 处理打印数字 print int |
| void mips\_printint(int index,int funcnum) | 处理打印字符 print char |
| void mips\_scanint(int index,int funcnum) | 处理 scan a(a不是数组) ,a为int |
| void mips\_scanchar(int index,int funcnum) | 处理 scan a(a不是数组) ,a为char |
| void mips\_funcret(int index,int funcnum) | 处理op1=RET 将被调用函数的返回值赋给变量 |
| void mips\_bz(int index,int funcnum) | 上一个表达式为0(false)，跳转 |
| void mips() | 生成mips代码 |
| int main(void) | 主函数 |

### 调用依赖关系



### 符号表管理方案

符号表表项：



符号表：



### 存储分配方案 栈顶$sp（低地址）

|  |
| --- |
| 局部数据区 |
| 返回值 |
| 返回地址 |
| 上个函数栈底 |

栈底$fp（高地址）

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。

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

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。

### 四元式设计\*

|  |  |
| --- | --- |
| 操作种类 | 四元式（中间代码） |
| int常量定义 const int x=100; | const int x=100 |
| char常量定义 const char x=’a’; | const char x=’a’ |
| int 变量定义 int x; | var int x |
| char变量定义 char x; | var char x |
| int 数组定义 int a[100]; | var int a[100] |
| char数组定义 char a[100]; | var char a[100] |
| int 函数定义 int function(int x...) | int function() |
| char函数定义char function(int x...) | char function() |
| void函数定义void function(int x...) | void function() |
| 函数int参数定义 int function(int x...) | para int x |
| 函数char参数定义 int function(char x...) | para char x |
| 加法 result=op1+op2; | result = op1 + op2 |
| 减法 result = op1 - op2; | result = op1 - op2 |
| 乘法 result = op1 \* op2; | result = op1 \* op2 |
| 除法 result = op1 / op2; | result = op1 / op2 |
| 函数调用 function(100...); | call function |
| 函数传参 function(100,x); | push 100  push x |
| 接收函数返回值 x=function(100,x); | push 100  push x  call function  x=RET |
| 给变量赋值 x=y; | x=y |
| 给数组元素赋值 x[a]=y; | x[a]=y |
| 取数组元素的值 y=x[a]; | y=x[a] |
| 打印字符串 printf(“aaaaaaaaa”); | print “aaaaaaaa” |
| 打印整数 printf(100); | print 100 |
| 打印字符 printf(‘z’); | print ‘z’ |
| 读取整数 scanf(x); | scan int x |
| 读取字符 scanf(c); | scan char c |
| 输出换行 （为了让结果明显，在每个printf语句后都加了换行） | printline |
| 返回一个表达式的值 return(exp); | ret exp |
| 返回空值 return; | ret null |
| 设置标签 | set Label\_1 |
| 大于等于 x>=y | x>=y |
| 小于等于 x<=y | x<=y |
| 等于 x==y | x==y |
| 不等于 x!=y | x!=y |
| 大于 x>y | x>y |
| 小于x<y | x<y |
| 无条件跳转 | GOTO Label\_1 |
| 上一条中间代码不满足条件跳转 | BZ Label\_1 |
| 上一条中间代码满足条件跳转（实际上没用上这条中间代码） | BNZ Label\_1 |

### 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，而其中的几种中间代码比较复杂，这里着重介绍一下。

对于函数传参的中间代码：push para，需要设计一个参数栈，遇到函数传参，就将参数地址（是数字的化直接将数字）压栈，而遇到函数调用：call function，就先将栈中元素弹栈，即取实参所在地址的值，并拷贝到形参所在地址，直到参数个数满足。

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是另一个难点，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。

对于跳转的中间代码BZ、BNZ、GOTO需要访问上一条中间代码，根据上一条中间代码的类型来确定采用哪一种mips指令。

### 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。

（2）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

如：

x = 1 + 99 优化为-> x = 100

x = 13 \* 5 优化为-> x = 65

（3）复制传播

如果出现形如x = y的中间代码，y可能是标识符或是数字，则在之后的中间代码之中，使用y来代替x，要注意的是如果离开当前基本块或是x、y被重新赋值，则停止代替。如果x为产生中间代码时产生的临时变量，则之后可以删除x = y这条中间代码。

（4）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。

（5）DAG图法（最后未实现）

//dagpoint表示dag图中的一个点

struct dagpoint{

int op; //op为操作符，op值为1,2,3,4分别对应+,-,\*,/,初始为0

int no; //no节点的编号

dagpoint\* lchild; //本节点左子节点

dagpoint\*rchild; //本节点右子节点

dagpoint\*parent[]; //本节点的父节点(可能不止一个)

};

将所有的dagpoint都存在一个结构体数组中。

//point表示结点表中的一个表项

struct point{

dagpoint\* dag;//表项中的dag图节点

char \*id;//表项中的标识符

};

采用教材中的dag图导出中间代码的启发式算法进行优化。

## （6）全局寄存器分配（引用计数或着色算法）（未实现）；

## （7）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）（未实现）；

### 出错处理

（1）以下为错误类型及编码：

#define ZEROSTARTNUM\_ERROR 1 //数字有前零

#define ILEGALCH\_ERROR 2 //非法字符

#define SINGLEQ\_ERROR 3 //缺少单引号

#define DOUBLEQ\_ERROR 4 //缺少双引号

#define ASSIGN\_ERROR 5 //缺少赋值号

#define ID\_ERROR 6 //缺少标识符

#define LPAR\_ERROR 7 //缺少(

#define RPAR\_ERROR 8 //缺少)

#define LBRACE\_ERROR 9 //缺少{

#define RBRACE\_ERROR 10 //缺少}

#define LBRACKET\_ERROR 11 //缺少[

#define RBRACKET\_ERROR 12 //缺少]

#define FUNCTYPE\_ERROR 13 //缺少函数类型标识符(int char void)

#define STA\_ERROR 14 //不是语句

#define SEMICOLON\_ERROR 15 //缺少;

#define FACTOR\_ERROR 16 //不是因子

#define ARRLEN\_ERROR 17 //数组长度定义时错误

#define PLUSZERO\_ERROR 18 //0前有+ -

#define INTEGER\_ERROR 19 //缺少整数

#define CHAR\_ERROR 20 //缺少字符

#define TYPE\_ERROR 21 //缺少类型标识符(int char)

#define COLON\_ERROR 22 //缺少:

#define CONSTVALUE\_ERROR 2 3 //缺少常量

#define CASE\_ERROR 24 //缺少case

#define ADDSUB\_ERROR 25 //缺少+ -

#define STEP\_ERROR 26 //步长错误

#define REOP\_ERROR 27 //缺少关系运算符(< > <= >= == !=)

#define OUTOFBOUND\_ERROR 28 //数组越界

#define VARDEF\_ERROR 29 //变量定义位置错误

#define DEFAULT\_ERROR 30 //缺少default

#define OUTOFTAB\_ERROR 31 //符号表溢出

#define GLOBALNAME\_ERROR 32 //全局量、函数与全局量、函数不可重名

#define LOCALNAME\_ERROR 33 //局部常、变量重名

#define UNDEFID\_ERROR 34 //未定义标识符

#define VARIABLE\_ERROR 35 //应为char int变量

#define NOTARR\_ERROR 36 //应该数组的地方使用了非数组变量

#define HASRETFUNC\_ERROR 37 //应为有返回值函数

#define DIFFVAR\_ERROR 38 //循环控制变量前后不一

#define PARACOUNT\_ERROR 39 //参数数目错误

#define TYPECONFL\_ERROR 40 //类型不一致

1. 出错处理

词法分析、语法分析、语义分析出现错误时，调用error.h的errormesage()函数，根据错误类型进行报错，对于一些致命错误如符号表溢出，将直接终止程序；对于一些危险性较小的错误如类型错误（将int值赋给char变量），将报错但是并不进行跳读；对于其他程序，在进行报错后，还要根据错误发生的位置，调用skip()函数（可变参数函数）进行跳读。

## 三．操作说明

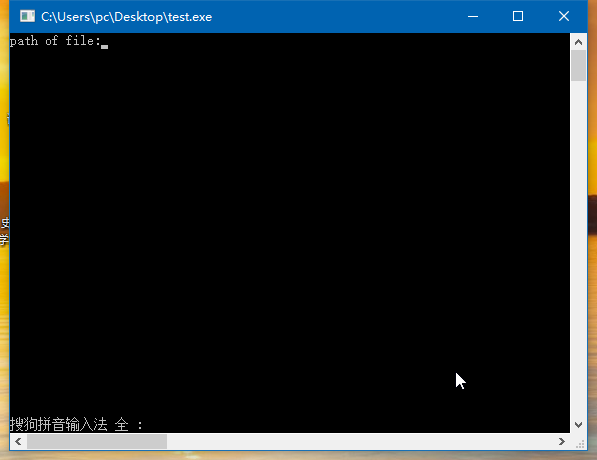
### 1．运行环境

运行环境：WINDOWS 10及以上

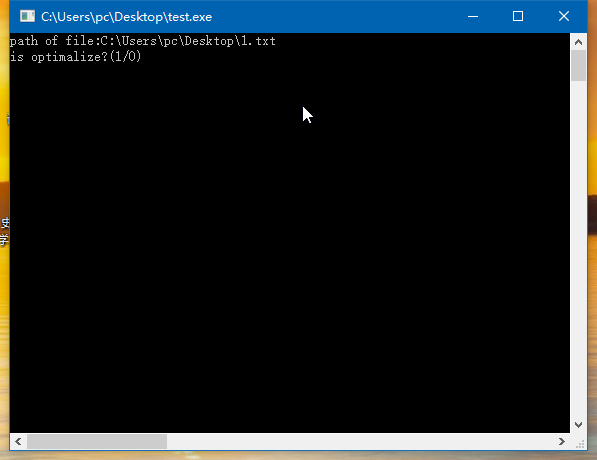
开发环境：Visual Studio10及以上

### 操作步骤

1. 双击test.exe或者启动工程，出现以下界面：



（2）输入C0代码的绝对路径:



（3）接着选择是否进行优化，输入1为进行优化，输入0为不进行优化

（4）最后在test.exe同一目录下将生成 grammar.txt（语法分析结果），error.txt （错误信息），midcode.txt（中间代码） ，mips.txt（目标代码）。

## 四．测试报告

### 1．测试程序及测试结果

正确程序5个

（1）

程序为：

int exp(int x,int y,char c){

int result;

result=0;

switch(c){

case '+':{;result=x+y;}

case '-':{;result=x-y;}

case '\*':{;result=x\*y;}

case '/':{;result=x/y;}

default:printf("not + - \* /");

}

return (result);

}

void main(){

printf(exp(1,exp(2,exp(3,exp(4,5,'/'),'-'),'\*'),'+'));

}

结果为：

7

（2）

程序为：（计算两个矩阵相交面积）

int max(int i,int j){

if(i>j){

return (i);

}

else{return (j);

}

}

int min(int i,int j){

if(i>j){

return (j);

}

else{

return (i);

}

}

void main(){

int Ax1,Ay1,Ax2,Ay2,Bx1,By1,Bx2,By2,a,b,MAX\_AX,MAX\_BX;

int MIN\_AX,MIN\_BX,MAX\_AY,MAX\_BY,MIN\_AY,MIN\_BY;

scanf(Ax1,Ay1,Ax2,Ay2);

scanf(Bx1,By1,Bx2,By2);

MAX\_AX=max(Ax1,Ax2);

MAX\_BX=max(Bx1,Bx2);

MIN\_AX=min(Ax1,Ax2);

MIN\_BX=min(Bx1,Bx2);

MAX\_AY=max(Ay1,Ay2);

MAX\_BY=max(By1,By2);

MIN\_AY=min(Ay1,Ay2);

MIN\_BY=min(By1,By2);

if(min(MAX\_AX,MAX\_BX)-max(MIN\_AX,MIN\_BX)>0){

if(min(MAX\_AY,MAX\_BY)-max(MIN\_AY,MIN\_BY)>0){

a=min(MAX\_AX,MAX\_BX)-max(MIN\_AX,MIN\_BX);

b=min(MAX\_AY,MAX\_BY)-max(MIN\_AY,MIN\_BY);

printf(a\*b);

}

}

else{

printf(0);

}

}

结果为：（>>代表输入）

>>-100

>>0

>>100

>>100

>>-200

>>50

>>150

>>250

10000

（3）

程序为：（求水仙花数）

void main(){

int i,j,k,m;

char s[10];

for(i=100;i<=999;i=i+1){

j=i/100;

k=(i-j\*100)/10;

m=(i-j\*100-k\*10);

if(j\*j\*j+k\*k\*k+m\*m\*m==i){

printf(i);

}

}

}

结果为：

153

370

371

407

（4）

程序为：

const int len=8;

const char t0='c',t1='o',t2='m', t3='p',t4='i',t5='l',t6='e',t7='r';

char s[8],temp;

int i,count;

int exp(int x,int y,char c){

int result;

result=0;

switch(c){

case '+':{;result=x+y;}

case '-':{;result=x-y;}

case '\*':{;result=x\*y;}

case '/':{;result=x/y;}

default:printf("not + - \* /");

}

return (result);

}

int function1(int x){

if(x<0){

x=exp(x,-1+0,'\*');

}

else{

x=exp(x,exp(x,0,'\*'),'-');

}

switch(x-exp(-0,x,'+')+x){

case 0: return (+0);

case 1: return (--1);

default: return (function1(x-2)+function1(x-1));

}

return (0);

}

char function2(){

char c;

c='Z';

count=exp(exp(1000,exp(count,1,'+'),'\*'),1000,'/');

scanf(c);

return (c);

}

void function3(int i){

printf("func3:",i);

}

void main(){

const int num=10;

int sum;

sum=0;

count=0;

temp='0';

printf(temp);

printf(t0);

printf(t1);

printf(t2);

printf(t3);

printf(t4);

printf(t5);

printf(t6);

printf(t7);

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

s[i]=function2();

}

printf("your input is:");

for(i=exp(len,-1,'+');i>=0;i=i-1){

printf(s[exp(len,i,'-')-1]);

}

printf("the amount of char is:",count);

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

switch(s[i]){

case 'c':printf(+function1(0));

case 'o':printf(+function1(0)\*function1(1));

case 'm':printf(+function1(1)+function1(2));

case 'p':printf(+function1(2)-function1(3));

case 'i':printf(+function1(3)/function1(4));

case 'l':printf(-function1(4)\*function1(5));

case 'e':printf(-function1(5)+function1(6));

case 'r':printf(-function1(6)-function1(7));

default:function3(i);

}

}

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

switch(exp(i,exp(count,0,'\*'),'+')){

case 3:printf("case 3:",i/3);

case 6:printf("case 6:",i/3);

case 9:printf("case 9:",i/3);

default:printf("default:",i);

}

}

for( i=len-1;i-0>=0;i=i-1){

if(s[i]>='g'){

if(s[i]!='g'){

sum=sum+s[i];

}

}

else{

sum=sum-s[i];

}

}

printf("const testa 01234",sum);

;;;

printf(exp(s[1]-0,exp(s[0]\*1,exp(function1(exp(function1(2),function1(3),'+')),exp(function1(4),function1(5),'\*'),'-'),'/'),'+'));

if(sum>30){

sum=sum-sum/30\*30;

}

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

count=exp(count,function1(i),'+');

}

printf(count);

return;

}

结果为：（>>为输入）

0

c

o

m

p

i

l

e

r

>>a>>b>>c>>d>>e>>f>>g>>hyour input is:

a

b

c

d

e

f

g

h

the amount of char is:8

func3:0

func3:1

0

func3:3

3

func3:5

func3:6

func3:7

default:0

default:1

default:2

case 3:1

default:4

default:5

case 6:2

default:7

default:8

case 9:3

const testa 01234-493

91

8

（5）

程序为：（冒泡排序）

void main()

{

int a[10];

int i, j, temp;

printf("Please input ten numbers:");

for (i = 0; i < 10; i=i+1) {

scanf(temp);

a[i]=temp;

}

for (i = 0; i < 9; i=i+1)

for (j = 0; j < 9 - i; j=j+1)

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

{

temp = a[j];

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

a[j+1] = temp;

}

}

for (i = 0; i < 10; i=i+1)

printf(a[i]);

return ;

}

结果为：

>>10

>>66

>>77

>>456

>>4563

>>89

>>-9

>>-77

>>-09

>>-765

-765

-77

-9

-9

10

66

77

89

456

4563

错误程序5个

（1）

程序为：

int exp(int x,int y,char c){

int result;

result=0;

switch(c){

case '+':{;result=x+y;}

case '-':{;result=x-y;}

case '\*':{;result=x\*y;}

case '/':{;result=x/y;}

:printf("not + - \* /");

}

return (result);

}

void main(){

printf(exp(1,exp(2,exp(3,exp(4,5,'/'),'-'),'\*'),'+'));

}

结果为：

(ERROR) line:9 char:3 lost a "default"

(ERROR) line:11 char:7 lost int/char/void at the head of a function

（2）

程序为：

int max(int i,int j){

if(i>j){

return (i);

}

else{return (j);

}

}

int min(int i,int j){

if(i>j){

return (j);

}

else{

return (i);

}

}

void main(){

int Ax1,Ay1,Ax2,Ay2,Bx1,By1,Bx2,By2,a,b,MAX\_AX,MAX\_BX;

char MIN\_AX,MIN\_BX,MAX\_AY,MAX\_BY,MIN\_AY,MIN\_BY;

scanf(Ax1,Ay1,Ax2,Ay2);

scanf(Bx1,By1,Bx2,By2);

MAX\_AX=max(Ax1,Ax2);

MAX\_BX=max(Bx1,Bx2);

MIN\_AX=min(Ax1,Ax2);

MIN\_BX=min(Bx1,Bx2);

MAX\_AY=max(Ay1,Ay2);

MAX\_BY=max(By1,By2);

MIN\_AY=min(Ay1,Ay2);

MIN\_BY=min(By1,By2);

if(min(MAX\_AX,MAX\_BX)-max(MIN\_AX,MIN\_BX)>0){

if(min(MAX\_AY,MAX\_BY)-max(MIN\_AY,MIN\_BY)>0){

a=min(MAX\_AX,MAX\_BX)-max(MIN\_AX,MIN\_BX);

b=min(MAX\_AY,MAX\_BY)-max(MIN\_AY,MIN\_BY);

printf(a\*b);

}

}

else{

printf(0);

}

}

结果为：

(WARNING) line:29 char:34 types are conflicting

(WARNING) line:29 char:41 types are conflicting

(WARNING) line:30 char:16 types are conflicting

(WARNING) line:30 char:23 types are conflicting

(WARNING) line:30 char:35 types are conflicting

(WARNING) line:30 char:42 types are conflicting

(WARNING) line:31 char:35 types are conflicting

(WARNING) line:31 char:42 types are conflicting

(WARNING) line:32 char:16 types are conflicting

(WARNING) line:32 char:23 types are conflicting

(WARNING) line:32 char:35 types are conflicting

(WARNING) line:32 char:42 types are conflicting

（3）

程序为：

void main(){

int i,j,k,m;

char s[10];

s=100;

for(i=100;i<=999;i=i+1){

j=i/100;

k=(i-j\*100)/10;

m=(i-j\*100-k\*10);

if(j\*j\*j+k\*k\*k+m\*m\*m==i){

printf(i);

}

}

结果为：

(ERROR) line:4 char:3 the ID is not defined

(ERROR) line:13 char:3 lost a '}'

（4）

程序为：

const int len=8;

const char t0='c',t1='o',t2='m', t3='p',t4='i',t5='l',t6='e',t7='r';

char s[8],temp

int i,count;

int exp(int x,int y,char c){

int result;

result=0;

switch(c)

case '+':{;result=x+y;}

case '-':{;result=x-y;}

case '\*':{;result=x\*y;}

case '/':{;result=x/y;}

default:printf("not + - \* /");

}

return (result);

}

int function1(int x){

if(x<0){

x=exp(x,-1+0,'\*');

}

else{

x=exp(x,exp(x,0,'\*'),'-')

}

switch(x-exp(-0,x,'+')+x){

case 0: return (+0;

case 1: return (--1);

default: return (function1(x-2)+function1(x-1));

}

return (0);

}

char function2(){

char c;

c='Z';

count=exp(exp(1000,exp(count,1,'+'),'\*'),1000,'/');

scanf(c);

return (c);

}

void function3(int i){

printf("func3:",i);

}

void main(){

const int num=10;

int sum;

sum=0;

count=0;

temp='0';

printf(temp);

printf(t0);

printf(t1);

printf(t2);

printf(t3);

printf(t4);

printf(t5);

printf(t6);

printf(t7);

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

s[i]=function2();

}

printf("your input is:");

for(i=exp(len,-1,'+');i>=0;i=i-1){

printf(s[exp(len,i,'-')-1]);

}

printf("the amount of char is:",count);

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

switch(s[i]){

case 'c':printf(+function1(0));

case 'o':printf(+function1(0)\*function1(1));

case 'm':printf(+function1(1)+function1(2));

case 'p':printf(+function1(2)-function1(3));

case 'i':printf(+function1(3)/function1(4));

case 'l':printf(-function1(4)\*function1(5));

case 'e':printf(-function1(5)+function1(6));

case 'r':printf(-function1(6)-function1(7));

default:function3(i);

}

}

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

switch(exp(i,exp(count,0,'\*'),'+')){

case 3:printf("case 3:",i/3);

case 6:printf("case 6:",i/3);

case 9:printf("case 9:",i/3);

default:printf("default:",i);

}

}

for( i=len-1;i-0>=0;i=i-1){

if(s[i]>='g'){

if(s[i]!='g'){

sum=sum+s[i];

}

}

else{

sum=sum-s[i];

}

}

printf("const testa 01234",sum);

;;;

printf(exp(s[1]-0,exp(s[0]\*1,exp(function1(exp(function1(2),function1(3),'+')),exp(function1(4),function1(5),'\*'),'-'),'/'),'+'));

if(sum>30){

sum=sum-sum/30\*30;

}

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

count=exp(count,function1(i),'+');

}

printf(count);

return;

}

结果为：

(ERROR) line:4 char:3 lost a ';'

(ERROR) line:9 char:6 lost a '{'

(ERROR) line:9 char:6 lost a '}'

(ERROR) line:24 char:2 lost a ';'

(ERROR) line:27 char:21 lost a ')'

(ERROR) line:49 char:6 the ID is not defined

(ERROR) line:50 char:13 the ID is not defined

(ERROR) line:50 char:14 lost a ')'

（5）

程序为：

void main()

{

int a[10];

int i, j, temp;

printf("Please input ten numbers:")

for (i = 0; i < 10; i=i+1) {

scanf(temp);

a[i]=temp;

}

for (i = 0; i < 9; i=i+1)

for (j = 0; j < 9 - i; j=j+1)

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

{

temp = a[j]

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

a[j+1] = temp;

}

for (i = 0; i < 10; i=i+1)

printf(a[i]);

return ;

}

结果为：

(ERROR) line:8 char:9 lost a ';'

(ERROR) line:18 char:19 lost a ';'

由于报错之后进行了跳读，所以跳读后可能出现虚报错误的情况，如错误样例1的第二条报错，如错误样例4的倒数2、3条报错，而跳读过度还有可能导致一部分错误被忽略，如最后一个程序中缺少}但是被跳过所以没有报错。

### 测试结果分析

覆盖情况：

正确样例：

（1）测试表达式计算、函数调用、输入输出、switch-case。

（2）测试表达式计算、函数调用、输入输出、if-else。

（3）测试表达式计算、函数调用、输入输出、if-else。

（4）所有情况均覆盖。

（5）测试输入输出、函数调用、for循环、if-else

错误样例：

（1）测试了缺少default的情况

（2）测试了变量类型冲突的情况

（3）测试了未定义标识符和缺少}的情况

（4）测试了缺少; { } )的情况

（5）测试缺少了; }的情况

## 五．总结感想

编译技术课设的单个工程代码量是迄今为止我学过的课程中最多的，课程的难度也可以说是迄今为止最难的。在编写代码时，我才发现课本上那些看上去自己已经懂了的知识点，到了自己编码实现时，竟然会不知从何下手。就拿生成目标代码这一阶段的编程来说，在写函数运行栈时，我感觉非常痛苦，因为实在搞不明白怎么把书上那个运行栈在mips架构下划分，我甚至在编译程序里编写了一个栈式结构作为函数运行栈，当我写完，我才突然明白运行栈不是编译程序中的一个数据结构，而指的是mips架构下的一段内存，我要做的就是确定这一段内存的上下界，之后我不得不将代码大改了一次。总之，编写一个编译程序真的帮助我充分理解了编译技术，这正应了“纸上得来终觉浅，绝知此事要躬行”。

除此之外，因为我的编译程序的代码超过了四千行，所以如果不保持良好的编程风格，代码的可维护性将极差，特别是在debug时，需要同时看着mips代码、中间代码、编译程序代码，还要进行单步调试，如果编译程序不将模块、结构划分清楚，无疑将给自己造成极大地困扰，这也算是让我们不得不规范自己的代码风格。

最后，感谢课程组的各位老师和助教，你们细心地为广大同学解疑答惑、设计测试程序，极大地帮助了同学们设计自己的编译程序。