**C0 编译器设计与实现**

小组成员

A

B

C

**目 录**

[一、题目：C0编译器的设计与实现 3](#_Toc59794895)

[二、总体设计 4](#_Toc59794896)

[三、模块划分 8](#_Toc59794897)

[四、词法分析 8](#_Toc59794898)

[五、语法分析 14](#_Toc59794899)

[六、语义分析 21](#_Toc59794900)

[七、C0编译程序语法错误处理的实现 26](#_Toc59794901)

[八、中间代码生成及目标代码生成 27](#_Toc59794902)

[九、解释器 29](#_Toc59794903)

[十、测试用例 33](#_Toc59794904)

[十一、心得感悟 39](#_Toc59794905)

**成员分工**

|  |  |
| --- | --- |
| **姓名** | **分工** |
| A | block(), statement(), gen()等核心函数编写、解释器interpreter的编写及整个系统的融合 |
| B | getch(), getsym(), initC0line, outPutFile()等函数编写，输入输出函数，词法分析 |
| C | loop(), condition(), expression(), term(), factor()等语法语义分析等函数的编写 |

# 一、题目：C0编译器的设计与实现

C0语言的语法结构定义如下：

<程序>->[<变量定义部分>] {<自定义函数定义部分>} <主函数>

<变量定义部分>-> int id {, id};

<自定义函数定义部分>-> ( int id | void id) '(' ')' <分程序>

<主函数>->void main'(' ')' <分程序>

<分程序>->'{' [<变量定义部分>] <语句序列> '}'

<语句序列>-><语句> {<语句>}

<语句>-> <条件语句>｜<循环语句> | '{'<语句序列>'}' | <自定义函数调用语句> | <赋值语句> | <返回语句> | <读语句> | <写语句> | ;

<条件语句>->if '('<表达式>')' <语句> [else <语句>]

<循环语句>->while '(' <表达式>')' <语句>

<自定义函数调用语句>-><自定义函数调用>;

<赋值语句>->id = <表达式>;

<返回语句>->return ['(' <表达式> ')'] ;

<读语句>->scanf '(' id ')';

<写语句>->printf '(' [ <表达式>] ')';

<表达式>-> [+｜-] <项> { (+｜-) <项>}

<项> -> <因子>｛(\*｜/) <因子>｝

<因子> -> id｜'(' <表达式>')' | num | <自定义函数调用>

<自定义函数调用>->id '(' ')'

其中，id代表标识符，num代表整数，其含义及构成方式与C语言相一致；C0源程序中的变量需先定义后使用，其作用域与生存期与C语言相一致；自定义函数可超前使用（调用在前，定义在后）。

根据上面给定的C0文法及其说明和下列定义的假想栈式指令系统，按递归下降分析法设计并实现该C0语言的编译器，生成栈式目标代码；编写栈式指令系统的解释执行程序，输出目标代码的解释执行结果。

要求所实现程序可带运行选项，以便对输出内容进行选择。所实现程序能分阶段依次输出下列内容：C0源程序及其编译结果（生成第一个函数目标代码前的符号表内容—通过选项选择、目标代码，出错信息）；解释执行目标代码，动态显示运行栈的内容—通过选项选择，输出执行结果。

假想的栈式指令系统表

|  |  |
| --- | --- |
| LIT 0 a | 将常数值取到栈顶，a为常数值 |
| LOD t a | 将变量值取到栈顶，a为相对地址，t为层数 |
| STO t a | 将栈顶内容送入某变量单元中，a为相对地址，t为层数 |
| CAL 0 a | 调用函数，a为函数地址 |
| INT 0 a | 在运行栈中为被调用的过程开辟a个单元的数据区 |
| JMP 0 a | 无条件跳转至a地址 |
| JPC 0 a | 条件跳转，当栈顶值为0，则跳转至a地址，否则顺序执行 |
| ADD 0 0 | 次栈顶与栈顶相加，退两个栈元素，结果值进栈 |
| SUB 0 0 | 次栈顶减去栈顶，退两个栈元素，结果值进栈 |
| MUL 0 0 | 次栈顶乘以栈顶，退两个栈元素，结果值进栈 |
| DIV 0 0 | 次栈顶除以栈顶，退两个栈元素，结果值进栈 |
| RED 0 0 | 从命令行读入一个输入置于栈顶 |
| WRT 0 0 | 栈顶值输出至屏幕并换行 |
| RET 0 0 | 函数调用结束后,返回调用点并退栈 |

# 二、总体设计

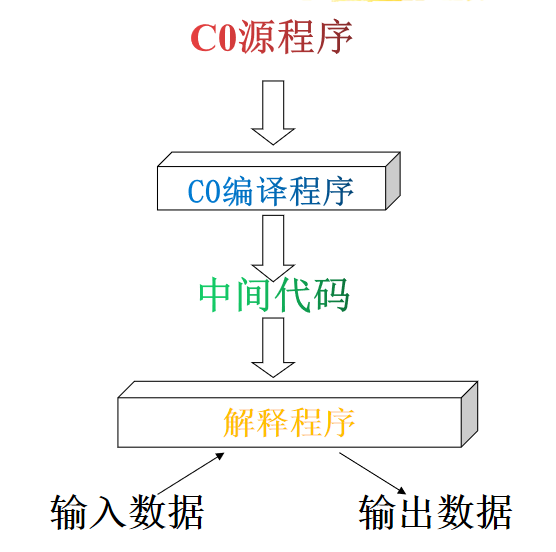
1.设计思路

（1）采用单趟方式实现编译。

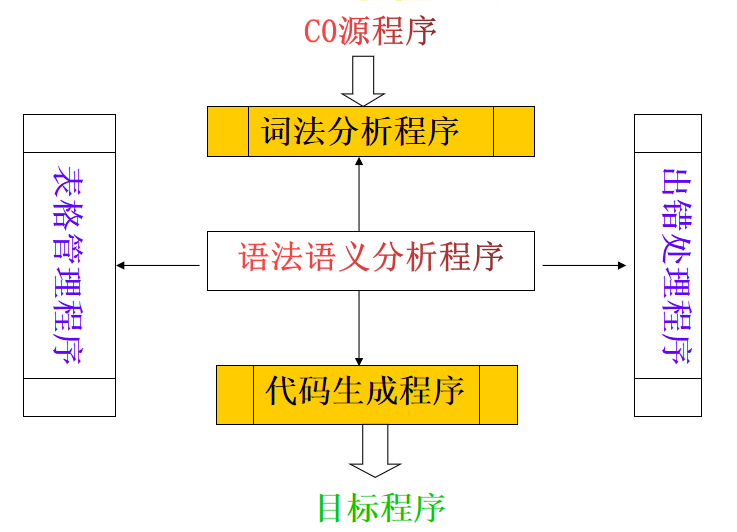
（2）以语法、语义分析程序为核心，词法分析程序和代码生成程序都作为一个过程，当语法分析需要读单词时就调用词法分析程序，而当语法、语义分析正确，需要生成相应的目标代码时，则调用代码生成程序。

（3）表格管理程序实现变量、常量和过程标识符的信息的登录和查找。

（4）出错处理程序，对词法和语法、语义分析遇到的错误给出源程序中出错的位置和与错误性质有关的编号，并进行错误恢复。

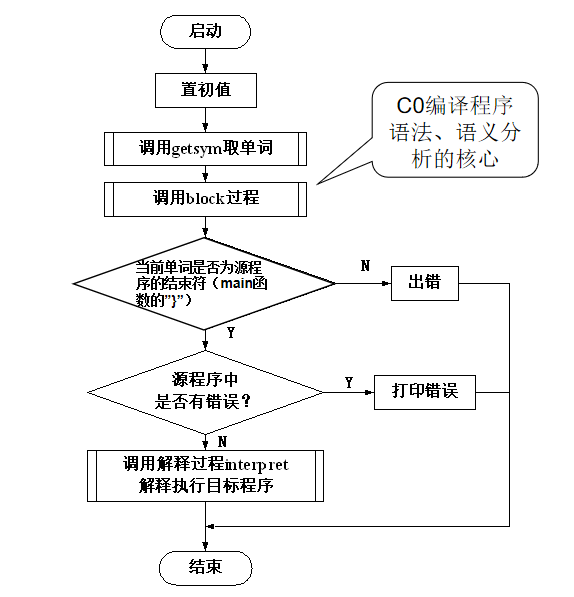


2.程序结构



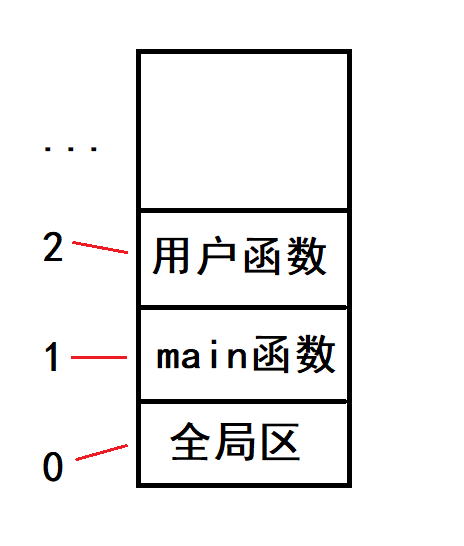
本次C0编译程序的结构基于教材中所提到的经典PL0结构进行设计和实现。

3.程序总体流程图



4.栈式设计

①假象栈式图



整体结构

以如下代码为例子：

int a,b;

int A(){

int c ,d ; ...

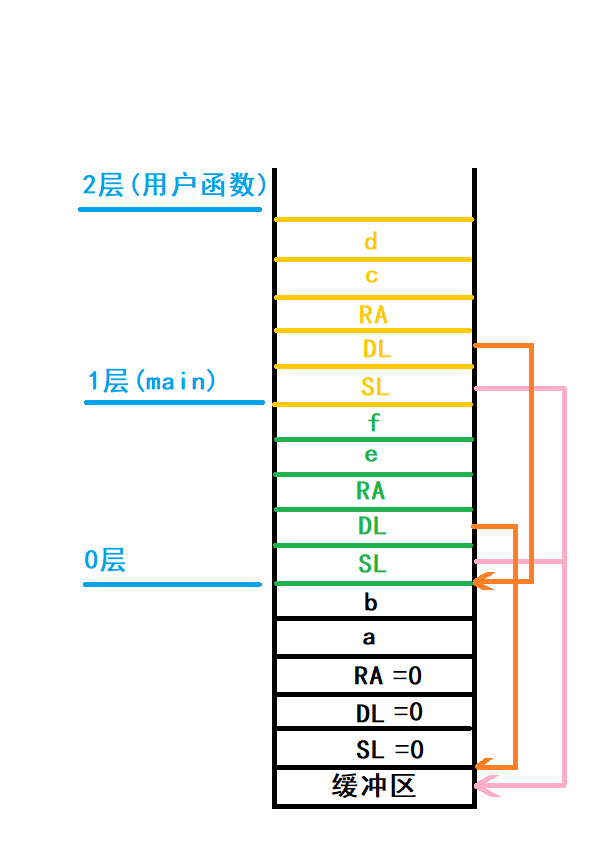
}

void main(){

int e ,f ; ...

}

那么栈结构如图：



注：

此处因为将main()函数放在第一层，因此进入主函数需要执行指令‘CAL 0 X’，才能正确进入主函数。

# 三、模块划分

编译程序完成从源程序到目标程序的翻译工作。理论上分为以下六个阶段：词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成，下面就这些方面，从除代码优化外的五个角度分别加以叙述。

# 四、词法分析

词法分析方面，主要涉及到两个函数：getch(), getsym().

1、基础函数getch()

|  |
| --- |
| public static int getch() {  *ch*=*line*[*cc*];  *cc*++;  return 0; } |

getch()的作用是读取储存在全局区line内部的单个字符，line存取用户输入的C0程序的全部单个字符形成的字符型数组。

2、核心函数 getsym()

|  |
| --- |
| public static int getsym() {  while(*ch*=='\n'||*ch*=='\r'||*ch*=='\t'||*ch*==' ')  *getch*();  if(*ch*>='a'&&*ch*<='z'||*ch*>='A'&&*ch*<='Z'||*ch*=='\_') *//检索名字或保留字* {  StringBuffer k=new StringBuffer();  do {  k.append(*ch*);  *getch*();  }while(*ch*>='a'&&*ch*<='z'||*ch*>='A'&&*ch*<='Z'||*ch*=='\_'||(*ch* >= '0'&&*ch* <= '9'));  *id*=k.toString();  *//搜索是否为保留字* if(*word*.contains(*id*))  *sym*=*wsym*[*word*.indexOf(*id*)];  else  *sym*=symbol.*ident*;  System.*out*.println(*sym*+": "+*id*);  }  else *//不是，则检索数字* {  if(*ch*>='0'&&*ch*<='9')  {  StringBuffer k=new StringBuffer();  *num*=0;  *sym*=symbol.*number*;  do {  k.append(*ch*);  *getch*();  }while(*ch*>='0'&&*ch*<='9');  *num*=Integer.*parseInt*(k.toString());  System.*out*.println(*sym*+" :"+*num*);  }  else *//不是，则按照单字符处理* {  *sym*=*ssym*(*ch*);  if(*sym*!=null)  System.*out*.println(*sym*);  else  *error*(2);  *getch*();  }  }  return 0; } |

getsym()忽略空格，利用getch()函数读取一个符号，读取的符号有以下几种情况：

|  |  |
| --- | --- |
| **单词类型** | **具体单词** |
| 保留字或关键字 | Int ,void ,main ,if ,else ,while ,return , scanf , printf |
| 运算符 | + ,- ,\* ,/ , = |
| 标识符 | 用户定义的变量名，常数名，过程名 |
| 常数 | 如1，2，3，67等整数 |
| 界符 | ’**,**’，’**;**’，’**(**‘，’**)**’，’**{**‘，’**}**’ |

（1）名字或者保留字以a-z,A-Z,’\_’开头的,后面可以是字母、’\_’和数字。只要读取到的符号不是字母数字之外的符号，且没超出程序最大长度的限制，则认为是一个单词。判断放入a数组的单词是关键字还是自定义的名字，找寻是否是保留字，如果找到，则把该关键字给word数组，否则，该单词即为不是关键字的标识符，将“ident”赋给sym。

（2）名字或者保留字以0-9的数字开头的单词是数字，固将number赋值给sym，由于getch()是一个字母一个字母的读，所以，我们需要计算出数字的值。

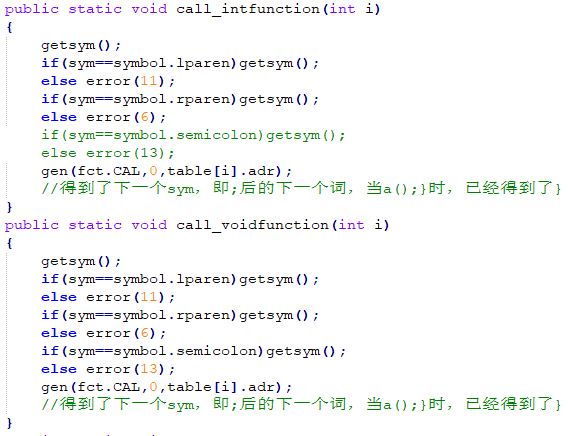
（3）既不是字母开头也不是数字开头的，则与符号进行匹配。有：（+、-、\*、/、（、）、{、}、,、;、=）,判断之后，将数组中与之对应的单词给sym。继续调用get\_ch()函数，获取下一个单词。

3、调用词法分析的方法：

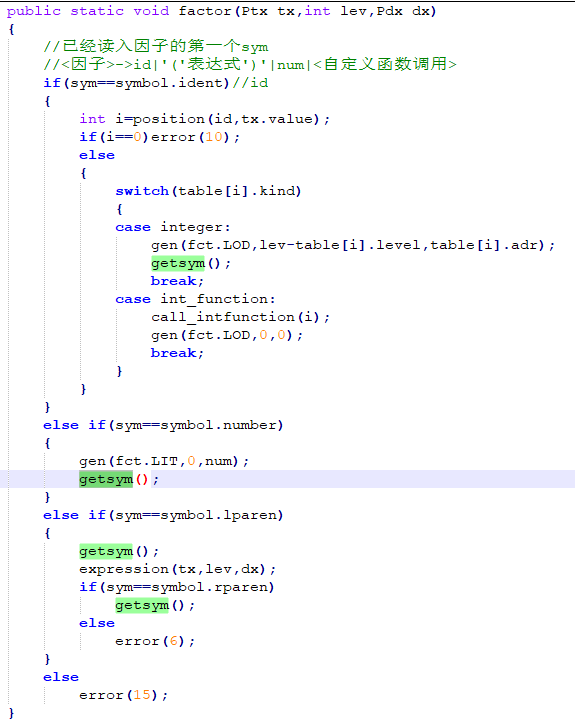
①语法分析中的 block() 函数：进行语法匹配，将调用词法分析判断当前符号是否为标识符



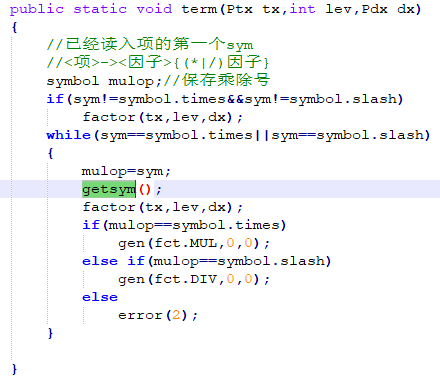
②call\_intfunction();call\_voidfunction()：这两个函数用来判断用户定义的函数类型，调用 getsym()判断是否缺少括号。



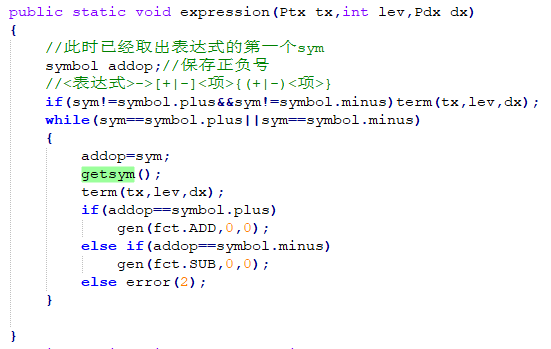
③factor()：用于语法分析中因子的判断。



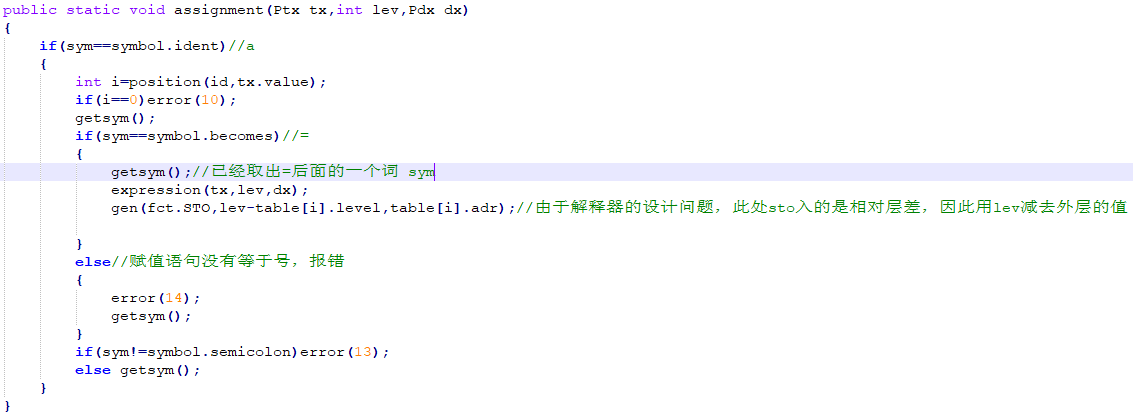
④term():用于语法分析中项的判断。



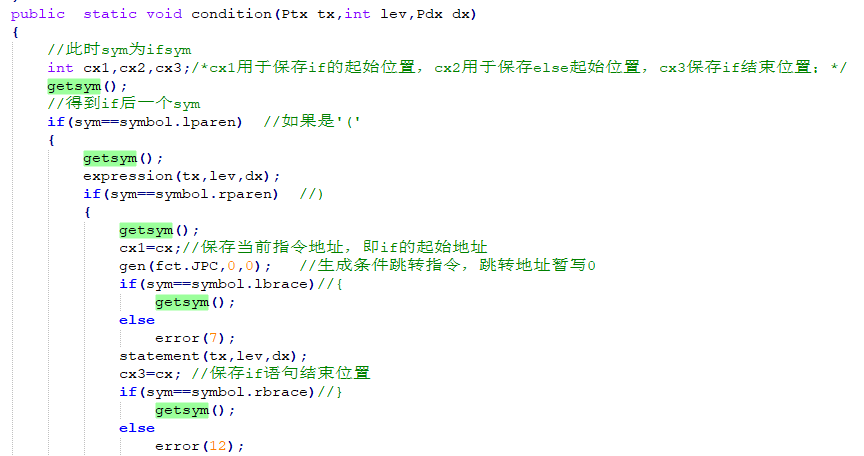
⑤expression():语法分析表达式的判断。



⑥assignment():赋值语句用于读取‘=’后的数据。



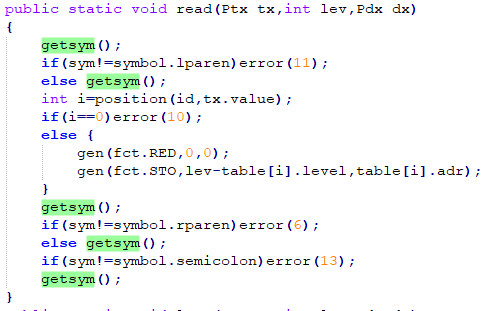
⑦condition():用于条件语句后的符号识别与报错。



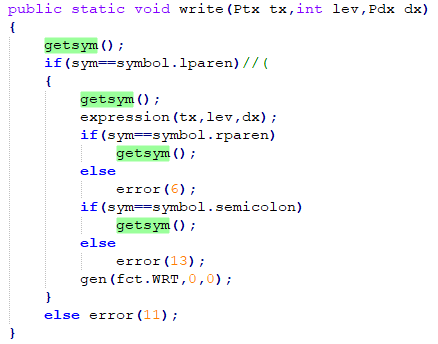
⑧loop():用于循环语句的括号读取和循环执行。



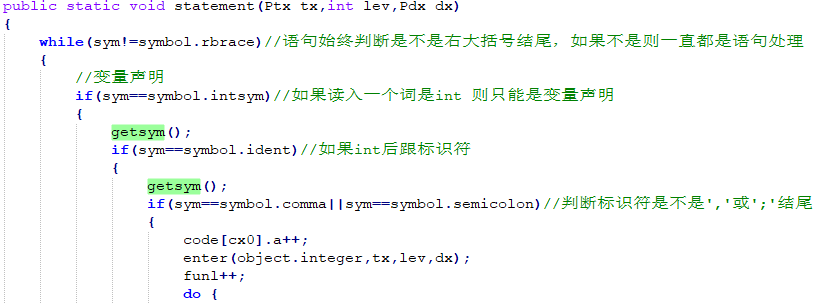
⑨read():指令“red”读取数字。



⑩write():指令“wrt”写。



⑪statement():处理语句，并申明错误。



# 五、语法分析

1.设计思路

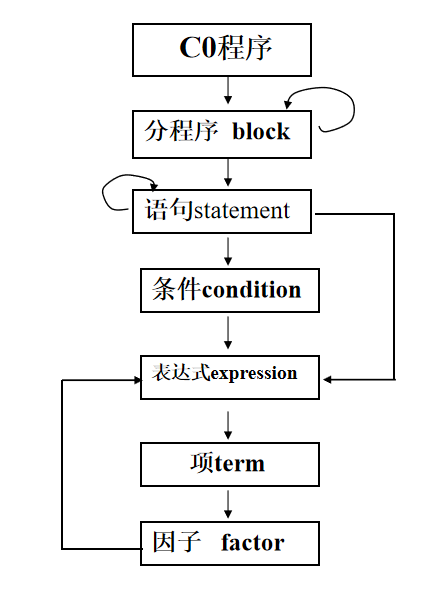
①自顶向下的语法分析

②递归子程序法（递归下降分析器recursive-descent parser）。

③对应每个非终结符（语法成分），编一个独立的处理子程序。

④由<程序>开始，按规则右部进行分析，遇到非终结符，则调用相应的处理过程，遇到终结符，则判断当前读入的单词是否与该终结符相匹配，若匹配，再读取下一个单词继续分析。

2.语法分析程序流程图



3.核心代码 block()

(1)执行条件

while(sym==symbol.voidsym||sym==symbol.intsym)

将符号 void 或者 int 作为代码分析的入口。根据题目条件，程序由int 函数/变量或者 void 函数作为入口。有关判断是那种类型入口的语句：

//int

if(sym==symbol.intsym)

…

if(sym==symbol.ident)//变量

…

else if(sym==symbol.lparen)//函数

…

//void

…

else if(sym==symbol.voidsym)//函数

…

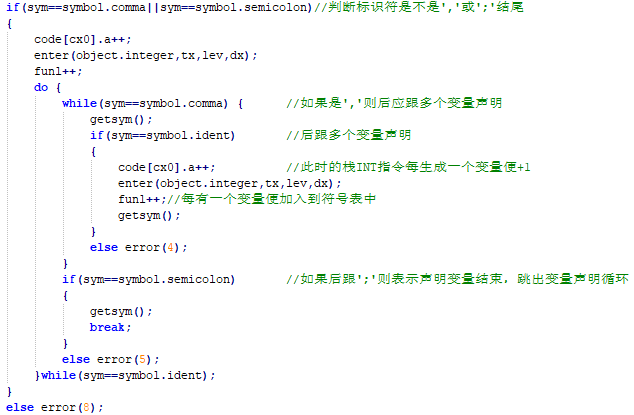
分析完毕后，递归执行block下的子程序。

(2)block下的子程序

①statement()：此函数用于处理用户定义函数部分开始条件:

while(sym!=symbol.rbrace)

//语句始终判断是不是右大括号结尾，如果不是则一直都是语句处理。当是以int开头，则为用户变量申明：



此处为用户开辟栈空间存储变量，并将处理层差。当是标识符开头时，可能是语句或者函数调用：



else if(sym==symbol.ifsym)

condition(tx,lev,dx);

else if(sym==symbol.whilesym)

loop(tx,lev,dx);

else if(sym==symbol.scanfsym)

read(tx,lev,dx);

else if(sym==symbol.printfsym)

write(tx,lev,dx);

else if(sym==symbol.returnsym)break;

else error(2);

不同标识符会指向不同子程序，如：

//准备按照条件语句处理；

//循环语句；

//返回语句处理。

②condition()：条件判断语句，入口为‘ifsym’。

int cx1,cx2,cx3;

cx1用于保存if的起始位置，cx2用于保存else起始位置，cx3保存if结束位置。条件语句用于中间代码生成的重要部分：

//if

getsym();

cx1=cx;//保存当前指令地址，即if的起始地址

gen(fct.JPC,0,0); //生成条件跳转指令，跳转地址暂写0

if(sym==symbol.lbrace)//{

getsym();

else

error(7);

statement(tx,lev,dx);

cx3=cx; //保存if语句结束位置

if(sym==symbol.rbrace)//}

getsym();

//else

…

if(sym==symbol.elsesym)

{

gen(fct.JMP,0,0);

getsym();

cx2=cx;

…

else

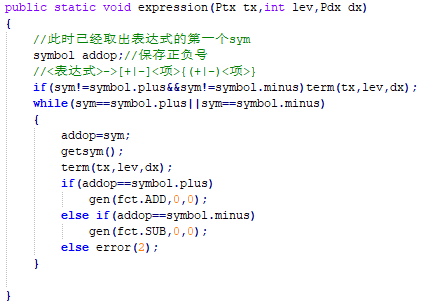
code[cx1].a=cx2;

code[cx3].a=cx;

}

cx2为else语句的起始地址，若有else语句，cx为语句执行完的位置则正是前面未定的跳转地址。这样就能实现中间代码的条件跳转和结束。

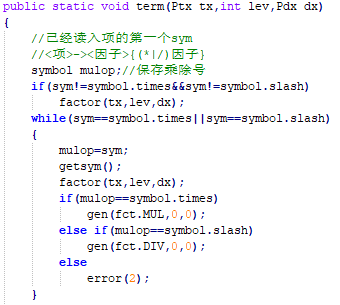
③expression()：表达式处理语句。



表达式由 + 和 - 分隔。使用getsym()进行加减匹配。

④term

项处理语句



项由乘除运算符分隔。使用getsym()进行乘除运算符匹配。

⑤factor()：因子处理语句

1）匹配到id 时：匹配到标识符id 时判断是函数还是整型变量。若函数则调用call\_intfuction()，若是整型则产生LOD命令。

if(sym==symbol.ident) {

int i=position(id,tx.value);

…

case integer:

gen(fct.LOD,lev-table[i].level,table[i].adr);

…

case int\_function:

call\_intfunction(i);

gen(fct.LOD,0,0);

2）匹配到number时：产生LIT命令。

else if(sym==symbol.number){

gen(fct.LIT,0,num);

getsym();}

3）匹配到左括号时：此时识别到左括号意味着又是可能的表达式，递归调用expression()。

else if(sym==symbol.lparen){…

expression(tx,lev,dx);

if(sym==symbol.rparen)

getsym();

…}

⑥loop()：循环语句处理，开始条件：

|  |
| --- |
| else if(*sym*==symbol.*whilesym*)*//循环语句* |

循环设计：

|  |
| --- |
| *getsym*(); cx1=*cx*;*//保存判断条件操作的位置 expression*(tx,lev,dx); cx2=*cx*;*//保存循环体的结束的下一位置 gen*(fct.*JPC*,0,0); |

循环结束：

|  |
| --- |
| if(*sym*==symbol.*rparen*)*//)* {  *getsym*();  if(*sym*==symbol.*lbrace*)*//{  getsym*();  else *error*(7);  *statement*(tx,lev,dx);  *gen*(fct.*JMP*,0,cx1);*//反填，重新判断条件  code*[cx2].a=*cx*;*//反填跳出循环的地址，与if类似* } |

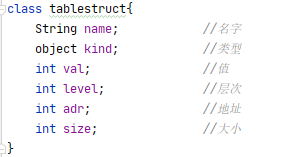
此处操作类似手动执行循环条件：不满足则在固定行间执行，满足条件则跳过循环语句执行紧接着的下条语句。

# 六、语义分析

1. 说明部分的分析与处理

（1）对每个过程（含主程序）说明的对象（变量，常量和过程）造符号表

建立符号表：



标识符类型，符号值

enum symbol:

|  |  |  |
| --- | --- | --- |
| nul | ident | number |
| lparen | rparen | becomes |
| rbrace | elsesym | ifsym |
| scanfsym | voidsym | whilesym |
| plus | minus | times |
| comma | semicolon | lbrace |
| intsym | printfsym | returnsym |
| mainsym | slash |  |

（2）登录标识符的属性。（包括种类，所在层次,值和分配的相对位置）

2.表格管理

|  |
| --- |
| *//写入符号表* public static void enter(object k,Ptx ptx,int lev,Pdx pdx)  {  ptx.value++;  *table\_num*++;  *table*[ptx.value].name=*id*;  *table*[ptx.value].kind=k;  switch(k){  case *integer*: *//对于整数，记录名字、类型、层以及相对位置，相对位置为该层的相对位置  table*[ptx.value].level=lev;  *table*[ptx.value].adr=pdx.value;  pdx.value++;  break;  case *int\_function*: *//对于函数，记录名字、类型和层  table*[ptx.value].level=lev;  *//new add  table*[ptx.value].adr=*cx*;  break;  case *void\_function*:  *table*[ptx.value].level=lev;  *table*[ptx.value].adr=*cx*;  break;  }  }  public static void outTable()   {  System.*out*.println("---------------名字表---------------------");  System.*out*.println("名字name\t类型kind\t值val\t层次level\t地址adr\t大小size\t"); *// String name; //名字 // object kind; //类型 // int val; //值 // int level; //层次 // int adr; //地址 // int size; //大小* for(int i=0;i<*table\_num*;i++)  {  System.*out*.println(*table*[i].name+"\t"+*table*[i].kind+"\t"+*table*[i].val+"\t"+*table*[i].level+"\t"+*table*[i].adr+"\t"+*table*[i].size+"\t");  }  } |

3.语句的分析与处理

核心函数：block()

|  |
| --- |
| public static int block() {  *//dx 该层下相对位置  //tx 名字表尾指针  //lev层  //cx 虚拟代码尾指针  cx0*=*cx*;*//cx0记录该层虚拟代码，用以记录函数返回值等，此处用以记录初始化数据的回填操作  gen*(fct.*INT*,0,4);  *gen*(fct.*JMP*,0,*enter\_main*);*//enter\_main用来记录main函数的位置，  enter\_main*=*cx*-1;  *dx*.value=4;  *//起始初始化 1单位隐藏单元用于return，3单元隐式单元 0,1,2,3* while(*sym*==symbol.*voidsym*||*sym*==symbol.*intsym*)  {  if(*sym*==symbol.*intsym*) *//如果是int，则可能是全局变量或函数* {  *getsym*();  if(*sym*==symbol.*ident*)  {  *getsym*();  if(*sym*==symbol.*comma*||*sym*==symbol.*semicolon*)*//定义数据* {  *code*[*cx0*].a++;  *enter*(object.*integer*,*tx*,*lev*,*dx*); *//插入符号表当中* do {  while(*sym*==symbol.*comma*) { *//如果是','则后应跟多个变量声明  getsym*();  if(*sym*==symbol.*ident*) *//后跟多个变量声明* {  *code*[*cx0*].a++; *//此时的栈INT指令每生成一个变量便+1  enter*(object.*integer*,*tx*,*lev*,*dx*); *//每有一个变量便加入到符号表中  getsym*();  }  else *error*(4);  }  if(*sym*==symbol.*semicolon*) *//如果后跟';'则表示声明变量结束，跳出变量声明循环* {  *getsym*();  break;  }  else *error*(5);  }while(*sym*==symbol.*ident*);  }  else if(*sym*==symbol.*lparen*)*//是函数* {  if(*lev*==0)*lev*=2;*//如果是在第零层，则直接等于2，让出第一层给main* else *lev*++; *//否则，层数应累次递增  dx*.value=3; *//初始化，每层起始相对位置为3，让出三个隐式单元  enter*(object.*int\_function*,*tx*,*lev*,*dx*);  *tx0*=*tx*.value;  *gen*(fct.*JMP*,0,*cx*+1);  *cx0*=*cx*;  *gen*(fct.*INT*,0,3);  *funl*=0;    *getsym*();  if(*sym*!=symbol.*rparen*)*error*(6);*//缺少右小括号* else *getsym*();  if(*sym*!=symbol.*lbrace*)*error*(7); *//缺少左大括号* else  {  *//此时的sym等于'{'，自定义函数部分开始：  getsym*();  *//得到'{'后下一个词  statement*(*tx*,*lev*,*dx*);  *//statement结束后，此时的sym应该是}* if(*sym*!=symbol.*returnsym*)*error*(16);*//有返回的函数类型最后一个返回回的应该是return* else  {  *getsym*();  *expression*(*tx*,*lev*,*dx*);*//表达式处理最后的return语句* }  *gen*(fct.*STO*,0,0);  if(*sym*!=symbol.*semicolon*)*error*(13);  else *getsym*();  if(*sym*!=symbol.*rbrace*)*error*(12);  else  *getsym*();  *gen*(fct.*RET*,0,0);  }   }  }   }  else if(*sym*==symbol.*voidsym*)  {  *getsym*();  if(*sym*==symbol.*ident*||*sym*==symbol.*mainsym*)*//只能是自定义void类型的函数* {  if(*sym*==symbol.*mainsym*)  {  *code*[*enter\_main*].a=*cx*;  *gen*(fct.*CAL*,0,*cx*+1);  *lev*=1;  }  else {  if(*lev*==0)*lev*=2;  else *lev*++;  }  *getsym*();  if(*sym*==symbol.*lparen*)  {  *dx*.value=3;  *enter*(object.*void\_function*,*tx*,*lev*,*dx*);  *tx0*=*tx*.value;  *gen*(fct.*JMP*,0,*cx*+1);  *cx0*=*cx*;  *gen*(fct.*INT*,0,3);  *funl*=0;      *getsym*();  if(*sym*!=symbol.*rparen*)*error*(6);  else *getsym*();  if(*sym*!=symbol.*lbrace*)*error*(7);  else  {  *getsym*();  *statement*(*tx*,*lev*,*dx*);  if(*sym*==symbol.*rbrace*)  *getsym*();  else  *error*(12);  *gen*(fct.*RET*,0,0);    }  }  else *error*(11);  }  }  }  return 1; } |

# 七、C0编译程序语法错误处理的实现

错误处理的原则：尽可能准确指出出错位置，错误性质，尽可能进行校正。通过错误代码的传递，在error()函数中指出过程中最早出现错误，错误表如下：

|  |  |
| --- | --- |
| **错误代码** | **详情** |
| 1 | C0File cannot open! |
| 2 | ident cannot be distinguished |
| 3 | Program too long, cannot generate instruct! table overflow |
| 4 | after ident variable's comma should be ident variable. |
| 5 | after ident variables should be semicolon. |
| 6 | lose ')' |
| 7 | lose '{'. |
| 8 | should be ',' or ';' |
| 9 | should be ident |
| 10 | Not Found ident. |
| 11 | lose '(' |
| 12 | lose '}'. |
| 13 | lose ';'. |
| 14 | assign statement lose '='. |
| 15 | factor error. |
| 16 | int\_function didn't define return type. |

# 八、中间代码生成及目标代码生成

1.实现结构

2.文件输入和输出

①输入

|  |
| --- |
| public static void initFile() {  System.*out*.println("请输入C0程序路径：");  Scanner input=new Scanner(System.*in*);  *C0File*=input.next(); } |

②输出中间代码和符号表

符号表

|  |
| --- |
| BufferedWriter out = new BufferedWriter(new FileWriter(outC0Path+"\\C0Table.txt")); out.write("名字name\t类型kind\t值val\t层次level\t地址adr\t大小size\n"); for(int i=1;i<=*tx*.value;i++) {  out.write(*table*[i].name+"\t"+*table*[i].kind+"\t"+*table*[i].val+"\t"+*table*[i].level+"\t"+*table*[i].adr+"\t"+*table*[i].size+"\n"); } out.close(); System.*out*.println("C0Table文件创建成功！"); } |

中间代码

|  |
| --- |
| BufferedWriter out = new BufferedWriter(new FileWriter(outC0Path+"\\C0MiddleProgram.txt")); for(int i=0;i<*cx*;i++) {  out.write(*code*[i].f+" "+*code*[i].l+" "+*code*[i].a+"\n"); } out.close(); System.*out*.println("C0MiddleProgram文件创建成功！"); } |

主函数调用

|  |
| --- |
| public static void main(String[] args) throws IOException {  *// TODO Auto-generated method stub  initFile*();  *init*();  System.*out*.println(*line*);  *getsym*();  *block*();  *test*();  *outPutFile*(); } |

# 九、解释器

|  |
| --- |
| #include <string> #include <iostream> #include<fstream> #include<sstream> using namespace std; enum fct {  *LIT*, *LOD*, *STO*, *CAL*, *INT*, *JMP*, *JPC*, *ADD*, *SUB*, *MUL*, *DIV*, *RED*, *WRT*, *RET* }; fct string\_to\_fct(string s) {  if (s == "LIT")  return *LIT*;  if (s == "LOD")  return *LOD*;  if (s == "STO")  return *STO*;  if (s == "CAL")  return *CAL*;  if (s == "INT")  return *INT*;  if (s == "JMP")  return *JMP*;  if (s == "JPC")  return *JPC*;  if (s == "ADD")  return *ADD*;  if (s == "SUB")  return *SUB*;  if (s == "MUL")  return *MUL*;  if (s == "DIV")  return *DIV*;  if (s == "RED")  return *RED*;  if (s == "WRT")  return *WRT*;  if (s == "RET")  return *RET*; } struct instruction {  enum fct f;  int l;  int a; }; const int codemaxsize = 500; const int stacksize = 50; instruction code[codemaxsize]; int base(int, int\*, int); void interpret(); int main() {  ifstream in("C0MiddleProgram.txt");  string filename;  string line;  string str1, str2, str3;  int instructionsum = 0;  if (in) *// 有该文件* {  while (getline(in, line)) *// line中不包括每行的换行符* {  cout << line << endl;  istringstream is(line);  is >> str1 >> str2 >> str3;*//已经处理完成。* code[instructionsum].f = string\_to\_fct(str1);  code[instructionsum].l = stoi(str2);  code[instructionsum].a = stoi(str3);  instructionsum++;  }  }  else *// 没有该文件* {  cout << "no such file" << endl;  }  for (int i = 0; i < instructionsum; i++)  {  cout<<"("<<i<<") "<<code[i].f<<" "<<code[i].l<<" "<<code[i].a<<endl;  }   interpret();  system("pause");  return 0; } void interpret() {  int p, b, t;  struct instruction i;  int s[stacksize];  cout << "start C0\n";  t = 0; b = 0; p = 0;  s[0] = s[1] = s[2] = 0;  do {  i = code[p];  p++;  switch (i.f)  {  case *LIT*:  s[t] = i.a;  t++;  break;  case *LOD*:  if(i.a==0&&i.l==0)  {  s[t]=s[0];  t++;  }  else  {  s[t] = s[base(i.l, s, b) + i.a];  t++;  }   break;  case *STO*:  t--;  if(i.a==0&&i.l==0)  {  s[0]=s[t];  }  else  s[base(i.l, s, b) + i.a] = s[t];  break;  case *CAL*:  s[t] = b;  s[t + 1] = 0;*//b->0* s[t + 2] = p;  b = t;  p = i.a;  break;  case *INT*:  t += i.a;  break;  case *JMP*:  p = i.a;  break;  case *JPC*:  t--;  if (s[t] == 0)  p = i.a;  break;  case *ADD*:  t--;  s[t - 1] = s[t] + s[t - 1];  break;  case *SUB*:  t--;  s[t - 1] = s[t - 1] - s[t];  break;  case *MUL*:  t--;  s[t - 1] = s[t - 1] \* s[t];  break;  case *DIV*:  t--;  s[t - 1] = s[t - 1] / s[t];  break;  case *RED*:  cin >> s[t];  t++;  break;  case *WRT*:  cout << s[t - 1]<<endl;  t--;  break;  case *RET*:  t = b;  p = s[t + 2];  b = s[t];  if (b == 0)  return;  break;  }  } while (p != 0); } int base(int l, int\* s, int b) {  int bt;   bt = b;  while (l > 0)  {  bt = s[bt];  l--;  }  return bt; } |

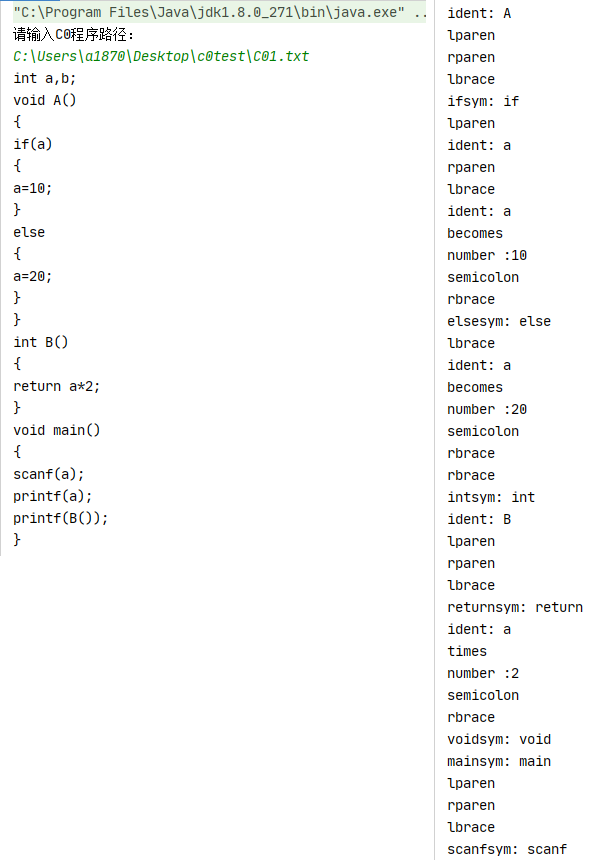
# 十、测试用例

①测试用例C01

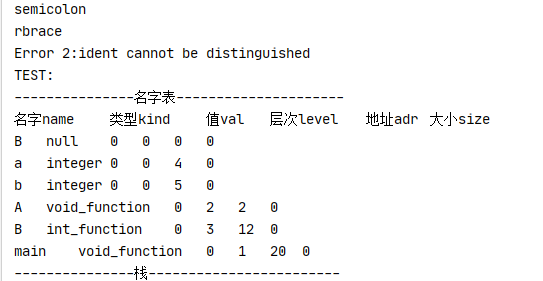
|  |
| --- |
| int a,b; void A(){  if(a){  a=10;}  else{  a=20;} } int B(){  return a\*2;} void main(){  scanf(a);  printf(a);  printf(B()); } |

在编译器下输入C01路径：“C:\Users\a1870\desktop\c0teset\C01.txt”

运行过程:

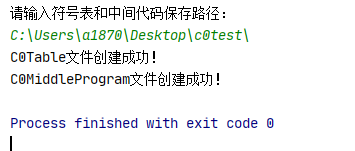


名字表：

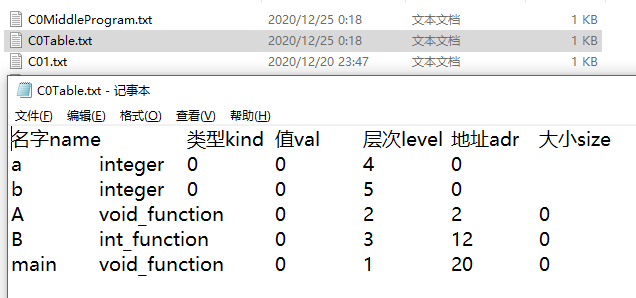


输出文件;

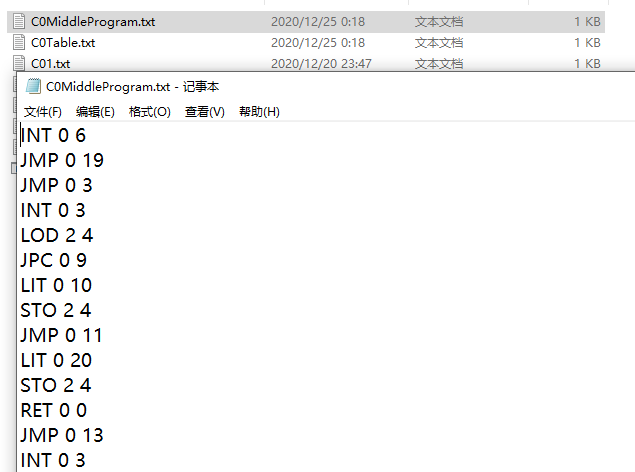
使用C++解释器



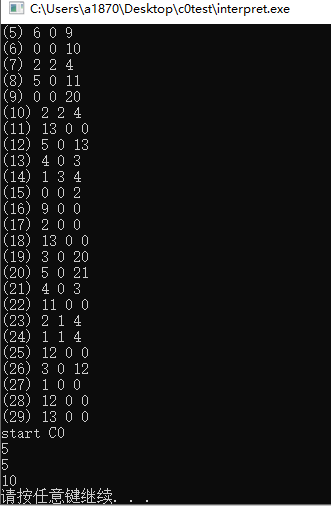
名字表.txt



中间代码.txt



使用C++编写的解释器interpret.exe



②C02

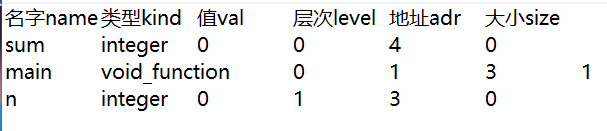
输入累加

|  |
| --- |
| int sum; void main(){  int n;  sum=0;  scanf(n);  while(n){  sum=sum+n;  n=n-1;}  printf(sum);} |

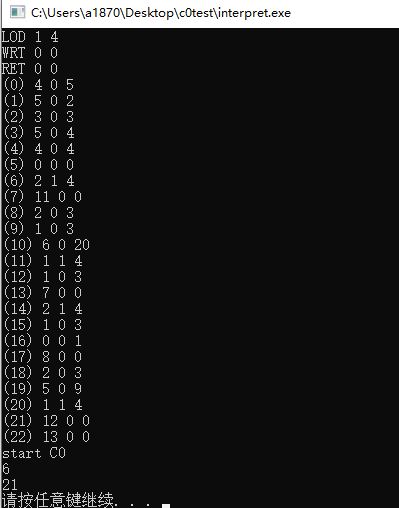
C02中间代码

|  |
| --- |
| INT 0 5 JMP 0 2 CAL 0 3 JMP 0 4 INT 0 4 LIT 0 0 STO 1 4 RED 0 0 STO 0 3 LOD 0 3 JPC 0 20 LOD 1 4 LOD 0 3 ADD 0 0 STO 1 4 LOD 0 3 LIT 0 1 SUB 0 0 STO 0 3 JMP 0 9 LOD 1 4 WRT 0 0 RET 0 0 |

C02名字表



运行结果：



③C03

用例

|  |
| --- |
| int a,b,c; void P(){  c=a+b;} void main(){  a=10;  scanf(b);  while(b){  P();  printf(2\*c);  scanf(b);} } |

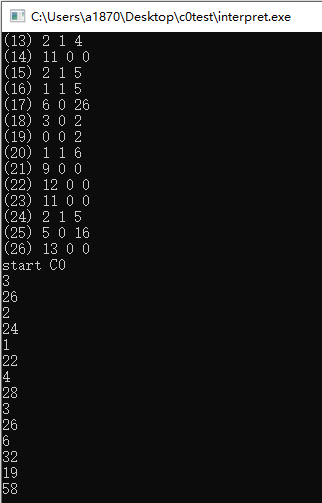
C03中间代码

|  |
| --- |
| INT 0 7 JMP 0 9 JMP 0 3 INT 0 3 LOD 2 4 LOD 2 5 ADD 0 0 STO 2 6 RET 0 0 CAL 0 10 JMP 0 11 INT 0 3 LIT 0 10 STO 1 4 RED 0 0 STO 1 5 LOD 1 5 JPC 0 26 CAL 0 2 LIT 0 2 LOD 1 6 MUL 0 0 WRT 0 0 RED 0 0 STO 1 5 JMP 0 16 RET 0 0 |

C03名字表



C03运行结果



# 十一、心得感悟

A：

这次大作业的小组实践，使我对编译原理有了更加深刻的认识。从一开始的理论学习，到仔细认真研学课本后PL/0的C语言编译实现代码，再到自己上手实践，编写C0语言的编译器和解释器，深切体会到了开发一个完完整整的高级语言编译器的不易。在最初，对编译的懵懂，转换为对任务毫无头绪的焦虑，但通过一遍又一遍的研究课本后的PL/0程序，和组内成员的交流，和其他组的成员的仔细探讨研究，最终搞清楚了总体开发逻辑，再次进行函数的划分、模块功能的划分，和组员合力开发出了编译器。这次的编译器开发大作业，是从理论入手，再到理论结合实践，最终在实践中去认识和发现、创新的过程，在这次过程中，有不知所措的焦虑，亦有当看到自己写好的程序通过单元测试、有理想的输出，和最后大作业整合完成的欣喜和雀跃。我很享受这次大作业的过程。

B：

这次大作业增强了我自己的动手能力和程序分析能力，同时也提高了和小组成员的合作能力。在此过程中，我体会到一个编译器工作的基本流程，在这过程中加深了对编译原理这门课的理解，我们通过查阅大量的资料，询问老师、同学，一步步实现编译器、解释器，也让我认识到：只有真的弄懂了，再去做才会让实验更顺利，不要盲目开始。在一开始我们小组的写作十分困难，毫无头绪，好在小组成员没有放弃共同努力，最终完成立编译器的编写。在过程中，我了解了词法分析器的实现过程。也体会到了编译的思想与实现过程。在实践中将理论与代码相结合，许多一知半解的地方也逐渐有了更加深刻的理解。

C：

团队的合作很重要，要多交流，不断地完善我们的代码，一起发现问题，解决问题。比如按照“从整体到局部再到整体的编程思路”，即先根据功能思考程序的整体结构，之后写出大体的数据、函数定义以及函数调用情况，之后再一步步进行细化和整合，最后再进行整体的测试、优化。而在小组合作的过程中，我也体会到了团队合作的重要性，团队中的合理分工不仅能简化工作、提高效率，也能促进成员之间相互学习、共同进步。