C0扩充文法编译器文档

学号： 10231016

姓名： 童 浩

2013年 1月 1日

1. **需求说明**

### 文法说明

* 1. **原始文法**

＜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＞‘]’ }  
＜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＞;｜＜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＞   ::=  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＞  
＜步长＞::= ＜无符号整数＞    
＜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＞‘)’]

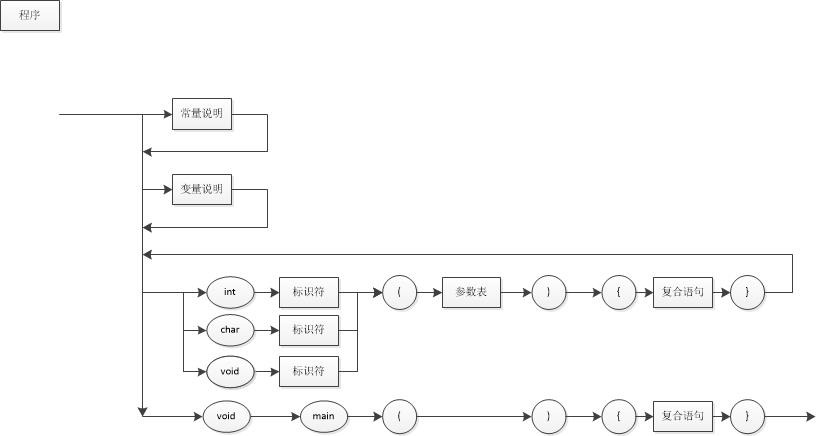
* 1. **文法的解读**

1. ＜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＞

＜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＞‘}’  
＜声明头部＞   ::=  int＜标识符＞|char＜标识符＞

＜主函数＞    ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’

＜参数＞    ::= ＜参数表＞

语法图为：

举例：

const int a=1,b=2;

int c;

void main()

{

return;

}

分析：

程序由可选的常量说明，变量说明，函数定义开始，且顺序已经限定好，由无返回值无参数的main函数结束。

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

语法图为：



范例：

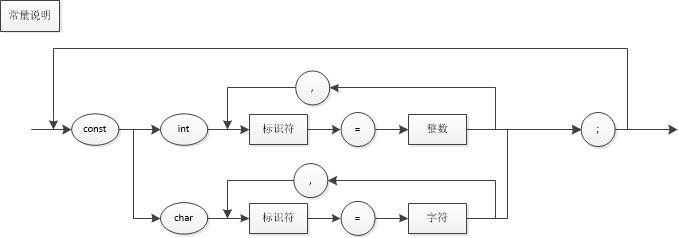
int a, char c, int b

分析：

参数表的文法规定了有（无）返回值函数说明的参数的格式，可以有0个或多个int或char类型的参数。

1. ＜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＞}

语法图为：



范例：

const int abc=1,bc=2;

const char a=’x’;

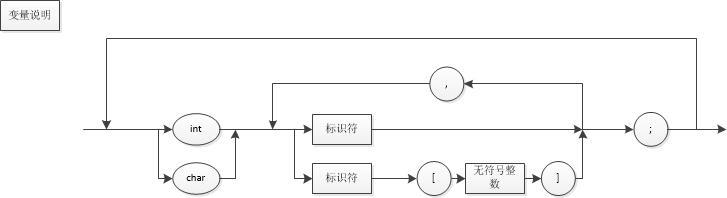
分析：

常量说明的文法规定了常量的说明格式。

1. ＜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＞‘]’ }

＜类型标识符＞      ::=  int | char

语法图为：



范例：

int a,b,c[50];

char d[20];

int e;

char f[10],g;

分析：

变量说明的文法规定了说明变量的格式。变量支持数组，说明类型顺序没有限定。

1. ＜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＞;  
   ＜条件语句＞  ::=  if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］

＜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＞

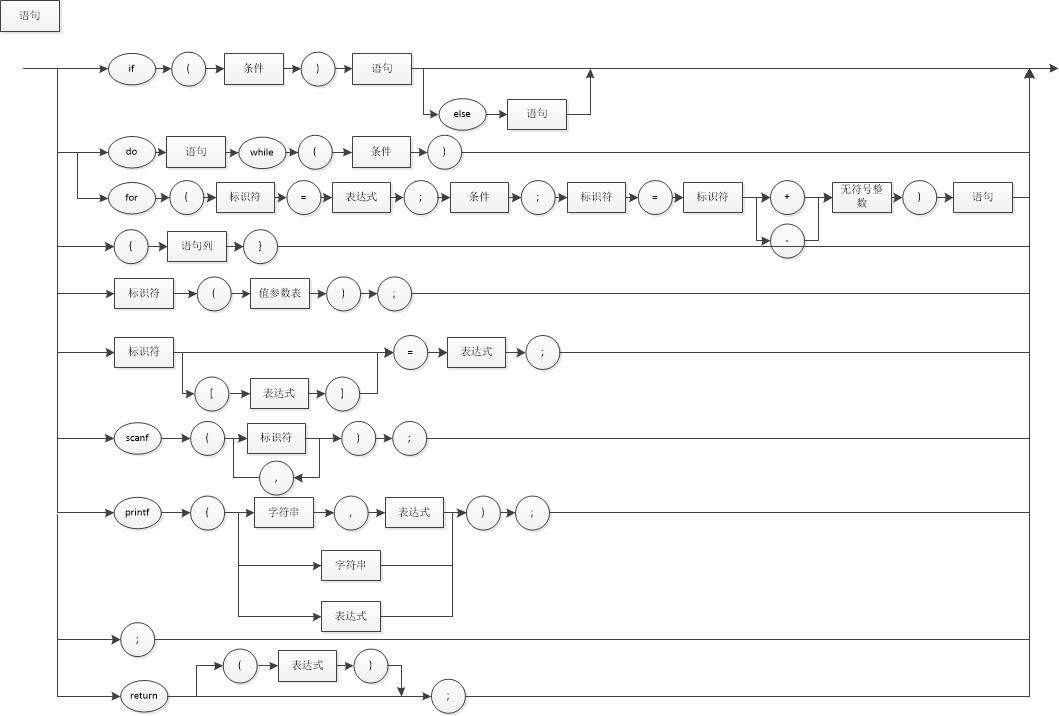
＜步长＞::= ＜无符号整数＞

＜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＞‘)’

＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞

＜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＞‘)’]

语法图为：



范例：

条件语句

if(a==b) a=1;

if(a==b) a=1;else a=2;

循环语句

do a=a+1; while(a<10)

for(i=1;i<10;i=i+1) a[i]=i;

{语句列}

{a=1;  
do a=a+1; while(a<10)}

{}

函数调用

gcd(3,5);

赋值语句

a=1;

a[10]=1;

a['a']=1;

a[i+1]=1;

读语句

scanf(a,b);

scanf(a);

写语句

printf("Hello World");

printf(a);

printf("Hello World",a+b);

空语句

;

返回语句

return;

return (a);

分析：

此部分主要说明了各种类型语句的结构组成，更进一步说就是说明了能够接受的语句语法格式；某些语句会有多种形式，在进行分析时应考虑全面。

1. ＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞

语法图为：



范例：

a,b,c+d/2,1

分析：

值参数表的文法规定了函数调用时的传参格式，可接受0个或多个表达式（变量，常量，整数等）。

1. ＜语句列＞   ::=｛＜语句＞｝

语法图为：



范例：

a=b;for(i=1;i<2;i++);

分析：

语句列的文法提供了多条语句实现的可能。

1. ＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞

语法图为：



范例：

const int a=1,b=2;

int var,sum;

char temp;

scanf(var,sum);

temp=var;

分析：

复合语句的文法规定了有返回值函数和无返回值函数以及主函数函数体的格式，可以有常量说明变量说明，必须有语句列。而且顺序不可更改。

1. ＜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  
   ＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==

语法图为：



范例：

a+b<=c

a+b-c

分析：

条件的文法规定了条件语句中条件的格式，可以是单个表达式，也可以是两个表达式通过比较运算符来比较。单个表达式情况下，表达式值为0则为假，否则为真。

1. ＜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＞‘)’

语法图为：



范例：

abc

a[20]

a[b+c]

20

'c'

gcd(20,74)

(abc)

分析：

此条语法图主要交代了因子的的详细组成，它有多种形式，编程时应考虑全面。

1. ＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}

＜乘法运算符＞  ::= \*｜/

语法图为：



范例：

x=x\*y;//x\*y既是项,也是表达式

分析：

此条文法比较简单，主要规范了项和因子的组成关系，并经常用于表达式；在程序中常是作为数学计算出现；

1. ＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

＜加法运算符＞ ::= +｜-

语法图为：



范例：

z:=x+y; /\*表达式的多种形式

z:=x-y; \*

z:=+5+x; \*

z:=-5+x; \*/

分析：

此处表述了表达式由项组成的具体方式，包含多种情况，值得注意；第一个项前面的+、-是表示如果第一个项需要是负数或正数，在数字前加上符号；后面的+、-基本上为运算符的+、-；

1. ＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝

语法图为：



范例：

123

1

10

分析：

无符号整数的文法规定了无前零整数的形式。

1. ＜整数＞        ::= ［＋｜－］＜无符号整数＞｜０

语法图为：



范例：

+1

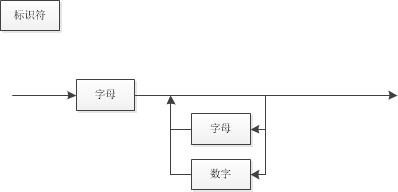
-10

0

分析：

整数的文法规定了整数的构成形式，可以有正负，但0前面不能有符号。

1. ＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

语法图为：

范例：

abc

abc0

分析：

标识符的文法规定了标识符的格式，必须以数字开头。

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

语法图为：



范例：

'a'

'+'

'9'

'\_'

分析：

字符的文法规定了字符的形式，必须用单引号，字符内容也有限定。

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

语法图为：



范例：

"Hello World"

分析：

字符串的文法规定了字符串的格式，必须有双引号，同时规定了字符串内容有所限定。

* 1. **属性翻译文法**

<AddOp>↑n::=+｜-

<MulOp>↑n::=\*｜/

<RelOp>↑n::=<｜<=｜>｜>=｜!=｜==

<Letter>↑n::=＿｜a｜．．．｜z｜A｜．．．｜Z

<Number>↑n /↑c↑s::=0｜<NzNumber>

<NzNumber>↑n /↑c↑s::=1｜．．．｜9

<Cha>↑c↑s::='<AddOp>'｜'<MulOp>'｜'<Letter>'｜'<Number>'

<Str>↑i::="｛十进制编码为32,33,35-126的ASCIICha｝"@insert

<Prog>::=［<ConstDec>］［<VarDec>］{<ValuedFuncDef>|<VoidFuncDef>}<MainFunc>

<ConstDec>::=const<ConstDef>;{const<ConstDef>;}

<ConstDef>::=int↑t <Identifier>↑n＝<Integer>↑c↑s @insert↓t↓n↓c↓s {,<Identifier>↑n＝<Integer>↑c↑s@insert↓t↓n↓c↓s }

| char↑t <Identifier>↑n＝<Cha>↑c↑s @insert↓t↓n↓c↓s {,<Identifier>↑n＝<Cha>↑c↑s @insert↓t↓n↓c↓s }

<UnInteger>::=<NzNumber>{<Number>}

<Integer>↑c↑s::=[+|-]<UnInteger>|0

<Identifier>↑n::=<Letter>{<Letter>|<Number>}

<DecHead>↑t↑n::=int↑t <Identifier>↑n

｜char↑t <Identifier>↑n

<VarDec>::=<VarDef>;{<VarDef>;}

<VarDef>::=<TypeId>↑t

(<Identifier>|<Identifier>‘[’<UnInteger>‘]’){,<Identifier>|<Identifier>‘[’<UnInteger>‘]’}

<TypeId>↑t::=int|char

<ValuedFuncDef>::=<DecHead>↑t↑n @begin↓t↓n↑i ‘(’<Para>↓n ‘)’ @stepin↓n@pop↓n ‘{’<ComStat>‘}’ @end @stepout

<VoidFuncDef>::=void<Identifier>↑n @begin↓n↑i ‘(’<Para>↓n ‘)’ @stepin↓n @ pop↓n ‘{’<ComStat>‘}’ @end @stepout

<ComStat>::=［<ConstDec>］［<VarDec>］<StatList>

<Para>↓n::=<ParaTable>↓n

<ParaTable>↓n::=<TypeId>↑t <Identifier>↑s@insert↓n↓t↓s {,<TypeId>↑t <Identifier>↑s @insert↓n↓t↓s }|<Empty>

<MainFunc>::=void main @begin ‘(’‘)’ @stepin ‘{’<ComStat>‘}’ @end @stepout

<Exp>↑i::=[+|-] ↑s <Term>↑i1{<AddOp>↑o <Term>↑i @add↓i1↓i2↓o↑i3} sign↓in↑i

<Term>↑i::=<Factor>↑i1 {<MulOp>↑o <Factor>@mul↓i1↓i2↓o↑i3}

<Factor>↑i::=<Identifier>↑n @find↓n↑i

｜<Identifier>‘[’<Exp>‘]’

｜<Integer>↑c↑s @ consttemp↓c↓s↑i

｜<Cha>↑c↑s @ consttemp↓c↓s↑i

｜<CallValuedFunc>↑i

｜‘(’<Exp>↑i ‘)’

<Stat>::=<CondStat>

｜<LoopStat>

｜‘{’<StatList>‘}’

｜<CallValuedFunc>;

｜<CallVoidFunc>;

｜<AssignStat>;

｜<ReadStat>;

｜<WriteStat>;

｜<Empty>;

｜<ReturnStat>;

<AssignStat>::=<Identifier>↑n @find↓n↑i1＝<Exp>↑i2 @assign↓i1↓i2

｜<Identifier>‘[’<Exp>‘]’=<Exp>

<CondStat>::=

if‘(’<Cond>↑i ‘)’ @ecbrf↓i↑a <Stat>@labprod↓a［else@cbrf↑b <Stat>@labprod↓b］

<Cond>↑i::=<Exp>↑i1<RelOp>↑o <Exp>↑i2@rel↓i1↓i2↓o↑i3

｜<Exp>↑i //Exp为0Cond为假，否则为真

<LoopStat>::=do@clabprod↑b <Stat>@brf↓b @labprod↓a while‘(’<Cond↑i >‘)’ @ecbrf↓i↑a

｜for‘(’<Identifier>↑n @find↓n↑i1＝<Exp>↑i2@assign↓i1↓i2@clabprod↑b;<Cond>↑i; ecbrf↓i↑a <Identifier>↑n1 @find↓n1↑i1＝<Identifier>↑n2 @find↓n2↑i2 (+|-)<Step>↑istep @add↓i1↓istep↓o↓i2 ‘)’<Stat>@brf↓b@labprod↓a

<Step>↑i::=<UnInteger>

<CallValuedFunc>↑j::=<Identifier>↑n @find↓n↑i ‘(’<ArgTable>↓i ‘)’ @call↓i↑j

<CallVoidFunc>::=<Identifier>↑n @find↓n↑i ‘(’<ArgTable>↓i ‘)’ @call↓i

<ArgTable>↓i::=<Exp>↑i1 @push ↓i↓i1↓m {,<Exp>↑i2 @push ↓i↓i2↓m }

｜<Empty>

<StatList>::=｛<Stat>｝

<ReadStat>::=scanf‘(’<Identifier>↑n @find↓n↑i @read↓i {,<Identifier>↑n @find↓n↑i @read↓i }‘)’

<WriteStat>::=printf‘(’<Str>↑i1,<Exp>↑i2 @write↓i1↓i2 ‘)’

｜printf‘(’<Str>↑i1 @write↓i1 ‘)’

｜printf‘(’<Exp>↑i2 @write↓i2 ‘)’

<ReturnStat>::=return[‘(’<Exp↑i @ret↓i >‘)’] @ret

### 目标代码说明

x86汇编代码

### 优化方案

* 1. 基本方案

基本块内的公共子表达式删除

全局寄存器分配

数据流分析、活跃变量分析

* 1. 实现算法

基本块划分

数据流分析

基本块内的公共子表达式删除

常量数值替换

全局寄存器分配

无用代码消除

栈空间压缩

### 开发环境

开发语言：C++

开发平台：Visual C++ 6.0

汇编工具：

Windows系统：Masm32v10

1. **详细设计**

### 程序结构

包含词法分析模块，语法分析模块，语义分析与中间代码生成模块，错误处理模块，中间代码优化模块，目标代码生成模块

### 类/方法/函数功能

* 1. **词法分析模块**
     1. **功能**
        1. 过滤掉程序中的空格、制表符以及换行符；
        2. 找到文法中定义的单词符号，识别出关键字；
        3. 将识别出的字符串表示整数转化为整数。
        4. 使编译器发现的错误信息与源程序的出错位置联系起来。
     2. **提供方法**

Token getSym();返回识别出的token

* 1. **语法分析模块**
     1. **功能**
        1. 按照文法，从源程序符号串中识别出各类语法成分；
        2. 进行语法检查，为语义分析和代码生成做准备。
     2. **算法**

递归子程序

* 1. **语义分析与中间代码生成模块**
     1. **功能**

根据属性翻译文法来确定源程序的意义，并生成四元式中间代码。

### 符号表结构

符号表类名为SymbolTable，符号表中的项目类名为Item。SymbolTable利用C++Vector类对Item实现封装，而Item的内部属性为：

int tag;//类型标签，与词法分析中tag一致，可由词法分析所得的token得到

int dimension;//维数，仅支持一维数组，所以最大为1。

string lexeme;//名称，在同一SymbolTable中唯一

string value;//值，对常量有效

int addr;//地址，对变量、函数有效

bool arr;是否为数组

SymbolTable\* pChildTable; //子符号表

SymbolTable\* pParentTable; //父母符号表

全局常量

全局变量

函数1

函数2

函数n

主函数

函数参数1

函数参数2

声明的常量

声明的变量

函数参数3

函数参数n

### 错误信息描述

* 1. **出错处理方案**

1. 在词法分析过程中输出非法token并定位
2. 在语法分析过程中输出不符合文法的错误定位
3. 在语义分析过程中重复定义变量以及其他的错误信息
   1. **错误信息编号和含义**

|  |  |
| --- | --- |
| **错误编号** | **含义** |
| 0 | 非法字符 |
| 1 | 单引号未配对 |
| 2 | 双引号未配对 |
| 3 | 标识符后缺少,或;或[ |
| 4 | 缺少函数名 |
| 5 | 常量不是在最先定义 |
| 6 | 缺少类型 |
| 7 | 缺少主函数 |
| 8 | 缺少const关键字 |
| 9 | 缺少分号 |
| 10 | 缺少标识符 |
| 11 | 缺少= |
| 12 | 常量重复定义 |
| 13 | 数组定义大小错误 |
| 14 | []未配对 |
| 15 | 非法生命变来那个 |
| 16 | 缺少( |
| 17 | ()未配对 |
| 18 | 缺少{ |
| 19 | {}未配对 |
| 20 | 非法使用变量 |
| 21 | 标识符未声明 |
| 22 | 未预料到的文件末端 |
| 23 | 非法表达式 |
| 24 | 有前零的数字 |
| 25 | 双零数字 |
| 26 | []未配对 |
| 27 | 缺少if |
| 28 | 缺少运算符 |
| 29 | 缺少while |
| 30 | 缺少printf |
| 31 | 无返回值函数有返回值 |
| 32 | 有返回值函数无返回值 |
| 33 | 非法字符串 |
| 34 | 标识符类型错误 |
| 35 | 数值类型错误 |
| 36 | 数据类型不匹配 |
| 37 | 变量重复声明 |
| 38 | 数组名被赋值 |
| 39 | 数组名被使用 |
| 40 | 数组引用超出范围 |
| 41 | 变量未被赋值而引用 |
| 42 | 调用未被定义的函数 |
| 43 | 函数参数不匹配 |

### 四元式设计

* 1. 四元式数据结构

class Quadruple{

public:

int Op;

Item\* Src1; //Item为符号表中的项目

Item\* Src2;

Item\* Obj;

//Methods emitted

}

* 1. 四元式定义

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作符 | 操作符定义值 | Src1 | Src2 | Obj | 语义 |
| ADD | 0 | Src1 | Src2 | Obj | 加法 |
| SUB | 1 | Src1 | Src2 | Obj | 减法 |
| MUL | 2 | Src1 | Src2 | Obj | 乘法 |
| DIV | 3 | Src1 | Src2 | Obj | 除法 |
| BT | 4 | Src1 | Src2 | Obj | 大于 |
| LT | 5 | Src1 | Src2 | Obj | 小于 |
| EQ | 6 | Src1 | Src2 | Obj | 等于 |
| BET | 7 | Src1 | Src2 | Obj | 大于或等于 |
| LET | 8 | Src1 | Src2 | Obj | 小于或等于 |
| NEQ | 9 | Src1 | Src2 | Obj | 不等于 |
| ASN | 10 | Src1 | NULL | Obj | 赋值 |
| EGTO | 11 | Src1 | NULL | Obj | 若为非则跳转 |
| RD | 12 | Src1 | NULL | NULL | 读入 |
| WR | 13 | Src1 | NULL | NULL | 写出 |
| LAB | 14 | Src1 | NULL | NULL | 设置标签 |
| GTO | 15 | NULL | NULL | Obj | 跳转 |
| BEGIN | 16 | Src1 | NULL | NULL | 函数头部 |
| APG | 17 | Src1 | NULL | NULL | 函数调用实参 |
| CALL | 18 | Src1 | NULL | NULL | 函数调用 |
| RET | 19 | NULL | NULL | NULL | 无返回值返回 |
| NRET | 20 | Src1 | NULL | Obj | 有返回值返回 |
| END | 21 | NULL | NULL | NULL | 函数尾部 |
| DINT | 22 | Src1 | NULL | NULL | int声明 |
| DCHR | 23 | Src1 | NULL | NULL | char声明 |
| REV | 24 | Src1 | NULL | Obj | 取负 |
| \_IN | 25 | Src1 | NULL | NULL | 函数定义形参 |
| ECALL | 26 | Src1 | NULL | NULL | 全局变量声明 |
| CST | 27 | Src1 | Src2 | Obj | 数组取值 |
| IDX | 28 | Src1 | Src2 | Obj | 数组取值 |
| IDXASN | 29 | Src1 | Src2 | Obj | 数组取值 |

### 语义分析

<AddOp>↑n::=+｜-

<MulOp>↑n::=\*｜/

<RelOp>↑n::=<｜<=｜>｜>=｜!=｜==

<Letter>↑n::=＿｜a｜．．．｜z｜A｜．．．｜Z

<Number>↑n /↑c↑s::=0｜<NzNumber>

<NzNumber>↑n /↑c↑s::=1｜．．．｜9

<Cha>↑c↑s::='<AddOp>'｜'<MulOp>'｜'<Letter>'｜'<Number>'

<Str>↑i::="｛十进制编码为32,33,35-126的ASCIICha｝"@insert

**void Str\_Insert(string n, Item\*& i); //将此项插入字符串Item数组,此数组由四元式管理器负责**

<Prog>::=［<ConstDec>］［<VarDec>］{<ValuedFuncDef>|<VoidFuncDef>}<MainFunc>

<ConstDec>::=const<ConstDef>;{const<ConstDef>;}

<ConstDef>::=int↑t <Identifier>↑n＝<Integer>↑c↑s @insert↓t↓n↓c↓s {,<Identifier>↑n＝<Integer>↑c↑s@insert↓t↓n↓c↓s }

| char↑t <Identifier>↑n＝<Cha>↑c↑s @insert↓t↓n↓c↓s {,<Identifier>↑n＝<Cha>↑c↑s @insert↓t↓n↓c↓s }

<UnInteger>::=<NzNumber>{<Number>}

**void ConstDef\_Insert(int t, string n, string c, int s); //向当前符号表插入全局常量，会执行相应检查,并生成四元式指令CST**

<Integer>↑c↑s::=[+|-]<UnInteger>|0

<Identifier>↑n::=<Letter>{<Letter>|<Number>}

<DecHead>↑t↑n::=int↑t <Identifier>↑n

｜char↑t <Identifier>↑n

<VarDec>::=<VarDef>;{<VarDef>;}

<VarDef>::=<TypeId>↑t(<Identifier>↑n@insert↓t↓n@alloc↓t |<Identifier>‘[’<UnInteger>‘]’){,<Identifier>|<Identifier>‘[’<UnInteger>‘]’}

**void VarDef\_Insert(int t, string n); //向当前符号表插入全局变量，会执行相应检查**

**void VarDef\_Alloc(int t, string n); //生成四元式指令DINT/DREA/DCHR**

<TypeId>↑t::=int|char

<ValuedFuncDef>::=<DecHead>↑t↑n @begin↓t↓n↑i ‘(’<Para>↓n ‘)’ @stepin↓n@pop↓n ‘{’<ComStat>‘}’ @end @stepout

**void FuncDef\_Begin(int t, string n, Item\*& i); //向当前符号表插入函数头部**

**void FuncDef\_Stepin(string n); //符号表压栈**

**void FuncDef\_Pop(string n); //保留**

**void FuncDef\_End(); //生成四元式指令END**

**void FuncDef\_Stepout(); //符号表出栈**

<VoidFuncDef>::=void<Identifier>↑n @begin↓n↑i ‘(’<Para>↓n ‘)’ @stepin↓n @ pop↓n ‘{’<ComStat>‘}’ @end @stepout

<ComStat>::=［<ConstDec>］［<VarDec>］<StatList>

<Para>↓n::=<ParaTable>↓n

<ParaTable>↓n::=<TypeId>↑t <Identifier>↑s@insert↓n↓t↓s {,<TypeId>↑t <Identifier>↑s @insert↓n↓t↓s }|<Empty>

**void ParaTable\_Insert(string n, int t, string s, int c); //向当前符号表插入形参**

<MainFunc>::=void main @begin ‘(’‘)’ @stepin ‘{’<ComStat>‘}’ @end @stepout

**void MainProc\_Begin(); //向当前符号表插入过程头部**

**void MainProc\_Stepin(); //符号表压栈**

**void MainProc\_End(); //生成四元式指令END**

**void MainProc\_Stepout(); //符号表出栈**

<Exp>↑i::=[+|-] ↑s <Term>↑i1{<AddOp>↑o <Term>↑i @add↓i1↓i2↓o↑i3} sign↓in↑i

**void Expr\_Add(Item\* i1, Item\* i2,int o, Item\*& res); //生成四元式指令ADD**

**void Expr\_Sign(Item\* in, Item\*& i); //生成四元式指令REV**

<Term>↑i::=<Factor>↑i1 {<MulOp>↑o <Factor>@mul↓i1↓i2↓o↑i3}

**void Term\_Mul(Item\* i1, Item\* i2,int o, Item\*& res); //生成四元式指令MUL**

<Factor>↑i::=<Identifier>↑n @find↓n↑i

｜<Identifier>‘[’<Exp>‘]’

｜<Integer>↑c↑s @ consttemp↓c↓s↑i

｜<Cha>↑c↑s @ consttemp↓c↓s↑i

｜<CallValuedFunc>↑i

｜‘(’<Exp>↑i ‘)’

**void Factor\_Find(string n, Item\*& i); //按名字查询符号表**

**void Factor\_ConstTemp(string c, int s, Item\*& i); //将此项插入Const temp Item数组,此数组由四元式管理器负责**

**void Factor\_Array\_Find(Item\* index,string n, Item\*& i);//查询数组的因子**

<Stat>::=<CondStat>

｜<LoopStat>

｜‘{’<StatList>‘}’

｜<CallValuedFunc>;

｜<CallVoidFunc>;

｜<AssignStat>;

｜<ReadStat>;

｜<WriteStat>;

｜<Empty>;

｜<ReturnStat>;

<AssignStat>::=<Identifier>↑n @find↓n↑i1＝<Exp>↑i2 @assign↓i1↓i2

｜<Identifier>‘[’<Exp>‘]’=<Exp>

**void AssignStat\_Find(string n, Item\*& i); //按名字查询符号表**

**void AssignStat\_Assign(Item\* i1, Item\* i2); //生成四元式指令ASN**

<CondStat>::=

if‘(’<Cond>↑i ‘)’ @ecbrf↓i↑a <Stat>@labprod↓a［else@cbrf↑b <Stat>@labprod↓b］

**void CtrlStat\_ECBrf(Item\* i,Item\*& a); //生成四元式指令EGOTO，创建标签Item在List Item数组，此数组由四元式管理器负责**

**void CtrlStat\_CBrf(Item\*& b); //生成四元式指令LAB，创建标签Item在List Item数组，此数组由四元式管理器负责**

**void CtrlStat\_Labprod(Item\* a); //生成四元式指令GOTO**

**void CtrlStat\_Brf(Item\* a); //生成四元式指令LAB**

**void CtrlStat\_CLabprod(Item\*& b); //生成四元式指令GOTO，创建标签Item在List Item数组，此数组由四元式管理器负责**

<Cond>↑i::=<Exp>↑i1<RelOp>↑o <Exp>↑i2@rel↓i1↓i2↓o↑i3

｜<Exp>↑i //Exp为0Cond为假，否则为真

**void LogExpr\_Rel(Item\* i1, Item\* i2, string o, Item\*& res); //生成四元式指令BT/LT/EQ/BET/LET/NEQ**

<LoopStat>::=do@clabprod↑b <Stat>@brf↓b @labprod↓a while‘(’<Cond↑i >‘)’ @ecbrf↓i↑a

｜for‘(’<Identifier>↑n @find↓n↑i1＝<Exp>↑i2@assign↓i1↓i2@clabprod↑b;<Cond>↑i; ecbrf↓i↑a <Identifier>↑n1 @find↓n1↑i1＝<Identifier>↑n2 @find↓n2↑i2 (+|-)<Step>↑istep @add↓i1↓istep↓o↓i2 ‘)’<Stat>@brf↓b@labprod↓a

**void CtrlStat\_ECBrf(Item\* i,Item\*& a); //生成四元式指令EGOTO，创建标签Item在List Item数组，此数组由四元式管理器负责**

**void CtrlStat\_CBrf(Item\*& b); //生成四元式指令LAB，创建标签Item在List Item数组，此数组由四元式管理器负责**

**void CtrlStat\_Labprod(Item\* a); //生成四元式指令GOTO**

**void CtrlStat\_Brf(Item\* a); //生成四元式指令LAB**

**void CtrlStat\_CLabprod(Item\*& b); //生成四元式指令GOTO，创建标签Item在List Item数组，此数组由四元式管理器负责**

**void LoopStat\_Find(string n, Item\*& i); //按名字查询符号表**

**void LoopStat\_Assign(Item\* i1, Item\* i2); //生成四元式指令ASN**

**void LoopStat\_Add(Item\* i1, Item\* istep, int o, Item\* i2);**

<Step>↑i::=<UnInteger>

<CallValuedFunc>↑j::=<Identifier>↑n @find↓n↑i ‘(’<ArgTable>↓i ‘)’ @call↓i↑j

**void FuncCall\_Find(string n, Item\*& i); //按名字查询符号表**

**void FuncCall\_Call(Item\* i, Item\*& j); //生成四元式指令ECALL**

<CallVoidFunc>::=<Identifier>↑n @find↓n↑i ‘(’<ArgTable>↓i ‘)’ @call↓i

**void ProcCall\_Find(string n, Item\*& i); //按名字查询符号表**

**void ProcCall\_Call(Item\* i); //生成四元式指令CALL**

<ArgTable>↓i::=<Exp>↑i1 @push ↓i↓i1↓m {,<Exp>↑i2 @push ↓i↓i2↓m }

｜<Empty>

**void ArguTable\_Check(Item\* i, Item\* i1, int m); //保存实参地址，并检查**

**void ArguTable\_ReversePush(Item\* i, Item\* item[], int n); //生成四元式指令ARG**

<StatList>::=｛<Stat>｝

<ReadStat>::=scanf‘(’<Identifier>↑n @find↓n↑i @read↓i {,<Identifier>↑n @find↓n↑i @read↓i }‘)’

**void ReadStat\_Find(string n, Item\*& i); //按名字查询符号表**

**void ReadStat\_Read(Item\* i); //生成四元式指令RD**

<WriteStat>::=printf‘(’<Str>↑i1,<Exp>↑i2 @write↓i1↓i2 ‘)’

｜printf‘(’<Str>↑i1 @write↓i1 ‘)’

｜printf‘(’<Exp>↑i2 @write↓i2 ‘)’

**void WriteStat\_Write(Item\* i); //生成四元式指令WR**

**void WriteStat\_Write(Item\* i1, Item\* i2); //生成四元式指令WR**

<ReturnStat>::=return[‘(’<Exp↑i @ret↓i >‘)’] @ret

**void RetStat\_NRet(Item\* i); //生成四元式指令NRET**

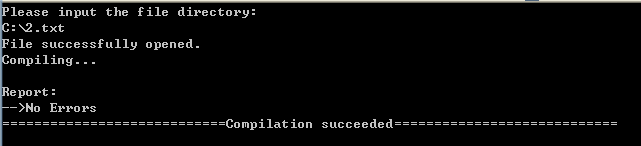
**void RetStat\_Ret(); //生成四元式指令RET**

1. **操作说明**

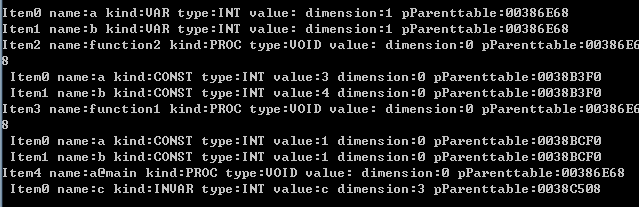
无需安装。在命令行下运行程序（推荐）或直接通过可执行文件打开程序，提示输入文件路径。输入正确路径后，点击回车，会进行编译。如有错误会输出错误。如没有错误会继续输出中间代码和优化后的中间代码，自动生成汇编码并连接。在软件中会提示是否要运行程序，输入y即运行。

例如：

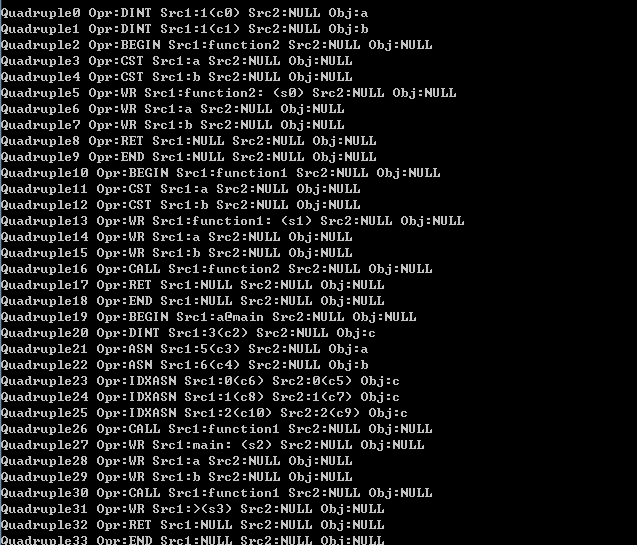
1. 编译信息



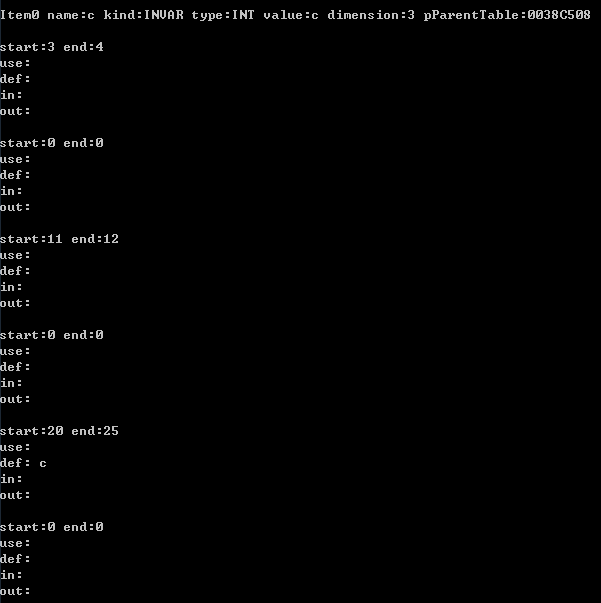
1. 符号表



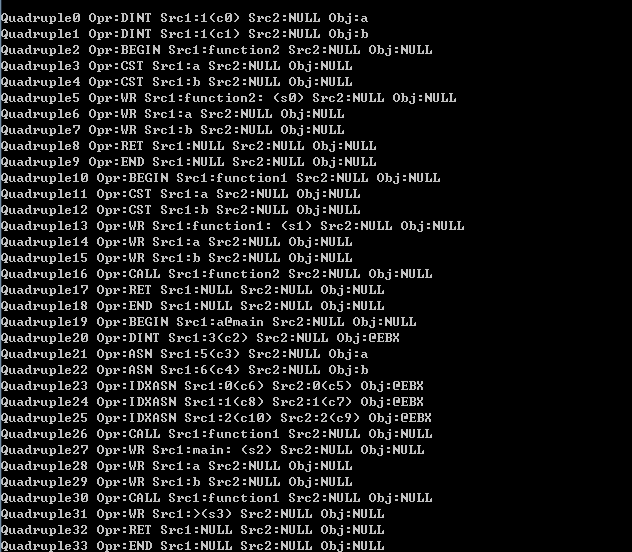
1. 中间代码



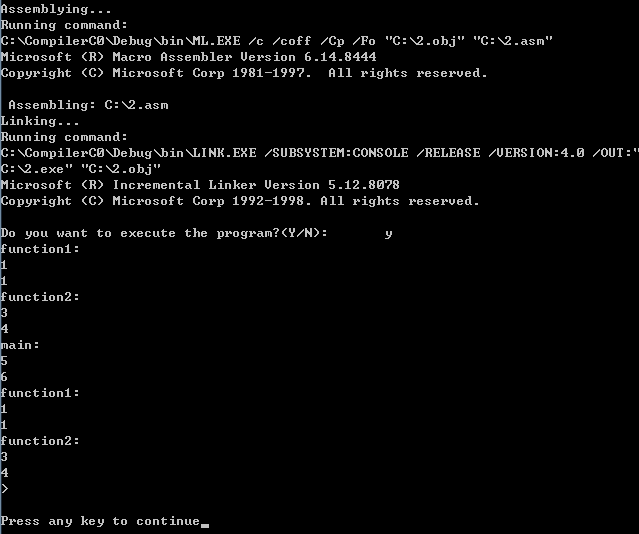
1. 数据流分析



1. 优化后的中间代码



1. 结果



1. **测试报告**

## 测试样例一（正确）

## 测试样例

int Fibonacci(int n){

if(n==0)

return (1);

if(n==1)

return (1);

else

return (Fibonacci(n-2)+Fibonacci(n-1));

}

void main(){

int k;

printf("Please input the Fabonacci number you want to know:");

scanf(k);

k=Fibonacci(k);

printf(k);

scanf(k);

return;

}

## 程序输出

输入一个数字，输出其斐波那契数

## 目标代码

. .386

.model flat,stdcall

option casemap:none

include C:\CompilerC0\Debug\include\windows.inc

include C:\CompilerC0\Debug\include\kernel32.inc

include C:\CompilerC0\Debug\include\msvcrt.inc

includelib C:\CompilerC0\Debug\lib\msvcrt.lib

includelib C:\CompilerC0\Debug\lib\kernel32.lib

include C:\CompilerC0\Debug\macros\macros.asm

.data

\_strs0 byte 'Please input the Fabonacci number you want to know:',0ah,0

.code

\_Fibonacci proc

push ebp

mov ebp,esp

sub esp,100h

mov eax,dword ptr [ebp+08h]

cmp eax,0

jne @label0\_1

mov dword ptr [ebp-04h],1

jmp @label0\_2

@label0\_1:

mov dword ptr [ebp-04h],0

@label0\_2:

cmp dword ptr [ebp-04h],0

jz @l0

mov eax,1

add esp,100h

pop ebp

ret 04h

@l0:

mov eax,dword ptr [ebp+08h]

cmp eax,1

jne @label1\_1

mov dword ptr [ebp-08h],1

jmp @label1\_2

@label1\_1:

mov dword ptr [ebp-08h],0

@label1\_2:

cmp dword ptr [ebp-08h],0

jz @l1

mov eax,1

add esp,100h

pop ebp

ret 04h

jmp @l2

@l1:

mov eax,dword ptr [ebp+08h]

sub eax,2

mov ebx,eax

mov eax,ebx

push eax

call \_Fibonacci

mov dword ptr [ebp-010h],eax

mov eax,dword ptr [ebp+08h]

sub eax,1

mov ebx,eax

mov eax,ebx

push eax

call \_Fibonacci

mov dword ptr [ebp-018h],eax

mov eax,dword ptr [ebp-010h]

add eax,dword ptr [ebp-018h]

mov ebx,eax

mov eax,ebx

add esp,100h

pop ebp

ret 04h

@l2:

add esp,100h

pop ebp

ret 04h

\_Fibonacci endp

start:

push ebp

mov ebp,esp

sub esp,100h

invoke crt\_printf,offset \_strs0

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-04h]

mov eax,dword ptr [ebp-04h]

push eax

call \_Fibonacci

mov dword ptr [ebp-08h],eax

mov eax,dword ptr [ebp-08h]

mov dword ptr [ebp-04h],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-04h]

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-04h]

add esp,100h

pop ebp

invoke ExitProcess,NULL

end start

执行结果

正确

## 测试样例二（正确）

## 测试样例

void main()

{

int i,n,sum,sum2,temp;

printf("Please enter the total count:");

scanf(n);

sum=0;

sum2=0;

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

{

scanf(temp);

sum=sum+temp;

sum2=sum2+temp\*temp;

}

printf("Sum=");

printf(sum);

printf("Avg=");

printf(sum/n);

printf("Var=");

printf(sum2/n-(sum/n)\*(sum/n));

return;

}

## 程序输出

先输入数n，回车，再输入n个数，用回车分隔，最后显示这n个数的和，平均数和方差。

## 目标代码

.386

.model flat,stdcall

option casemap:none

include C:\CompilerC0\Debug\include\windows.inc

include C:\CompilerC0\Debug\include\kernel32.inc

include C:\CompilerC0\Debug\include\msvcrt.inc

includelib C:\CompilerC0\Debug\lib\msvcrt.lib

includelib C:\CompilerC0\Debug\lib\kernel32.lib

include C:\CompilerC0\Debug\macros\macros.asm

.data

\_strs0 byte 'Please enter the total count:',0ah,0

\_strs1 byte 'Sum=',0ah,0

\_strs2 byte 'Avg=',0ah,0

\_strs3 byte 'Var=',0ah,0

.code

start:

push ebp

mov ebp,esp

sub esp,100h

invoke crt\_printf,offset \_strs0

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-08h]

mov eax,0

mov dword ptr [ebp-0ch],eax

mov eax,0

mov dword ptr [ebp-010h],eax

mov eax,0

mov dword ptr [ebp-04h],eax

@l0:

mov eax,dword ptr [ebp-04h]

cmp eax,dword ptr [ebp-08h]

jge @label0\_1

mov dword ptr [ebp-018h],1

jmp @label0\_2

@label0\_1:

mov dword ptr [ebp-018h],0

@label0\_2:

cmp dword ptr [ebp-018h],0

jz @l1

jmp @l2

@l3:

mov eax,dword ptr [ebp-04h]

add eax,1

mov dword ptr [ebp-04h],eax

jmp @l0

@l2:

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-014h]

mov eax,dword ptr [ebp-0ch]

add eax,dword ptr [ebp-014h]

mov edi,eax

mov eax,edi

mov dword ptr [ebp-0ch],eax

mov eax,dword ptr [ebp-014h]

mov ecx,dword ptr [ebp-014h]

imul ecx

mov esi,eax

mov eax,dword ptr [ebp-010h]

add eax,esi

mov ebx,eax

mov eax,ebx

mov dword ptr [ebp-010h],eax

jmp @l3

@l1:

invoke crt\_printf,offset \_strs1

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-0ch]

invoke crt\_printf,offset \_strs2

mov eax,dword ptr [ebp-0ch]

cdq

mov ecx,dword ptr [ebp-08h]

idiv ecx

mov dword ptr [ebp-028h],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-028h]

invoke crt\_printf,offset \_strs3

mov eax,dword ptr [ebp-010h]

cdq

mov ecx,dword ptr [ebp-08h]

idiv ecx

mov dword ptr [ebp-02ch],eax

mov eax,dword ptr [ebp-0ch]

cdq

mov ecx,dword ptr [ebp-08h]

idiv ecx

mov dword ptr [ebp-030h],eax

mov eax,dword ptr [ebp-0ch]

cdq

mov ecx,dword ptr [ebp-08h]

idiv ecx

mov esi,eax

mov eax,dword ptr [ebp-030h]

mov ecx,esi

imul ecx

mov ebx,eax

mov eax,dword ptr [ebp-02ch]

sub eax,ebx

mov dword ptr [ebp-03ch],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-03ch]

add esp,100h

pop ebp

invoke ExitProcess,NULL

end start

执行结果

正确

## 测试样例三（正确）

## 测试样例

int \_a;

int getCharByInt(int i){

int sum;

sum = 0;

do{

i = i + 1;

sum = sum + i;

}while(i<=\_a)

return (sum);

}

void main()

{

int sum;

\_a = 100;

scanf(sum);

printf(getCharByInt(sum));

}

## 程序输出

输入一个数，计算从这个数之后一直加到101的值。

## 目标代码

.386

.model flat,stdcall

option casemap:none

include C:\CompilerC0\Debug\include\windows.inc

include C:\CompilerC0\Debug\include\kernel32.inc

include C:\CompilerC0\Debug\include\msvcrt.inc

includelib C:\CompilerC0\Debug\lib\msvcrt.lib

includelib C:\CompilerC0\Debug\lib\kernel32.lib

include C:\CompilerC0\Debug\macros\macros.asm

.data

\_\_a dword 1 dup(?)

.code

\_getCharByInt proc

push ebp

mov ebp,esp

sub esp,100h

mov eax,0

mov edi,eax

@l0:

mov eax,dword ptr [ebp+08h]

add eax,1

mov esi,eax

mov eax,esi

mov dword ptr [ebp+08h],eax

mov eax,edi

add eax,dword ptr [ebp+08h]

mov ebx,eax

mov eax,ebx

mov edi,eax

mov eax,dword ptr [ebp+08h]

cmp eax,\_\_a

jg @label0\_1

mov dword ptr [ebp-010h],1

jmp @label0\_2

@label0\_1:

mov dword ptr [ebp-010h],0

@label0\_2:

cmp dword ptr [ebp-010h],0

jz @l1

jmp @l0

@l1:

mov eax,edi

add esp,100h

pop ebp

ret 04h

\_getCharByInt endp

start:

push ebp

mov ebp,esp

sub esp,100h

mov eax,100

mov \_\_a,eax

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-04h]

mov eax,dword ptr [ebp-04h]

push eax

call \_getCharByInt

mov dword ptr [ebp-08h],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-08h]

add esp,100h

pop ebp

invoke ExitProcess,NULL

end start

## 执行结果

正确

## 测试样例四（正确）

## 测试样例

const char ch='a';

const int const1=2;

void main()

{

int list[10];

char list2[10];

int b1,b2;

scanf(b1,b2);

printf(b1);printf(ch+b2);

printf(" ");

for(b1=0;b1<10;b1=b1+1)

{

list[b1]=ch+b1;

list2[b1]=list[b1];

printf(list[b1]);

printf(list2[b1]);

printf(" ");

}

return;

}

## 程序输出

测试字符与整数的相加，并根据变量类型输出字符或对应的ansii码值。

## 目标代码

.386

.model flat,stdcall

option casemap:none

include C:\CompilerC0\Debug\include\windows.inc

include C:\CompilerC0\Debug\include\kernel32.inc

include C:\CompilerC0\Debug\include\msvcrt.inc

includelib C:\CompilerC0\Debug\lib\msvcrt.lib

includelib C:\CompilerC0\Debug\lib\kernel32.lib

include C:\CompilerC0\Debug\macros\macros.asm

.data

\_strs0 byte ' ',0ah,0

\_strs1 byte ' ',0ah,0

\_ch equ 'a'

\_const1 equ 2

.code

start:

push ebp

mov ebp,esp

sub esp,100h

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-054h]

invoke crt\_scanf,SADD("%d"),addr dword ptr [ebp-058h]

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-054h]

mov eax,'a'

add eax,dword ptr [ebp-058h]

mov dword ptr [ebp-05ch],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-05ch]

invoke crt\_printf,offset \_strs0

mov eax,0

mov dword ptr [ebp-054h],eax

@l0:

mov eax,dword ptr [ebp-054h]

cmp eax,10

jge @label0\_1

mov dword ptr [ebp-060h],1

jmp @label0\_2

@label0\_1:

mov dword ptr [ebp-060h],0

@label0\_2:

cmp dword ptr [ebp-060h],0

jz @l1

jmp @l2

@l3:

mov eax,dword ptr [ebp-054h]

add eax,1

mov dword ptr [ebp-054h],eax

jmp @l0

@l2:

mov eax,'a'

add eax,dword ptr [ebp-054h]

mov dword ptr [ebp-064h],eax

mov eax,dword ptr [ebp-054h]

mov ecx,4

imul ecx

add eax,ebp

mov edx,dword ptr [ebp-064h]

mov dword ptr [eax-028h],edx

mov eax,dword ptr [ebp-054h]

mov ecx,4

imul ecx

add eax,ebp

mov eax,dword ptr [eax-028h]

mov ebx,eax

mov eax,dword ptr [ebp-054h]

mov ecx,4

imul ecx

add eax,ebp

mov edx,ebx

mov dword ptr [eax-050h],edx

mov eax,dword ptr [ebp-054h]

mov ecx,4

imul ecx

add eax,ebp

mov eax,dword ptr [eax-028h]

mov dword ptr [ebp-06ch],eax

invoke crt\_printf,SADD("%d",0ah,0),dword ptr [ebp-06ch]

mov eax,dword ptr [ebp-054h]

mov ecx,4

imul ecx

add eax,ebp

mov eax,dword ptr [eax-050h]

mov dword ptr [ebp-070h],eax

invoke crt\_printf,SADD("%c",0ah,0),dword ptr [ebp-070h]

invoke crt\_printf,offset \_strs1

jmp @l3

@l1:

add esp,100h

pop ebp

invoke ExitProcess,NULL

end start

## 执行结果

正确

## 测试样例五（正确）

## 测试样例

int a,b,c,d;

void main(){

b = 20;

a = 4;

d = 1;

c = 0;

while (d != 0){

if ((10 \* d + 1) >= a \* 5){

c = c + 2;

a = a \* 2;

d = d - 4;

}

else

d = d + 5;

}

printf(c);

}

## 程序输出

屏幕输出67108918，此为溢出值

## 目标代码

.386

.model flat,stdcall

option casemap:none

include C:\RadASM\Masm32\include\windows.inc

include C:\RadASM\Masm32\include\kernel32.inc

include C:\RadASM\Masm32\include\msvcrt.inc

includelib C:\RadASM\Masm32\lib\msvcrt.lib

includelib C:\RadASM\Masm32\lib\kernel32.lib

include C:\RadASM\Masm32\include\macros.asm

.data

\_a dword ?

\_b dword ?

\_c dword ?

\_d dword ?

.code

start:

push ebp

mov ebp,esp

sub esp,100h

mov eax,20

mov \_b,eax

mov eax,4

mov \_a,eax

mov eax,1

mov \_d,eax

mov eax,0

mov \_c,eax

@l0:

mov eax,\_d

cmp eax,0

je @label0\_1

mov dword ptr [ebp-04h],1

jmp @label0\_2

@label0\_1:

mov dword ptr [ebp-04h],0

@label0\_2:

cmp dword ptr [ebp-04h],0

jz @l1

mov eax,10

mov ecx,\_d

imul ecx

mov edi,eax

mov eax,edi

add eax,1

mov esi,eax

mov eax,\_a

mov ecx,5

imul ecx

mov ebx,eax

mov eax,esi

cmp eax,ebx

jl @label1\_1

mov dword ptr [ebp-014h],1

jmp @label1\_2

@label1\_1:

mov dword ptr [ebp-014h],0

@label1\_2:

cmp dword ptr [ebp-014h],0

jz @l5

mov eax,\_c

add eax,2

mov edi,eax

mov eax,edi

mov \_c,eax

mov eax,\_a

mov ecx,2

imul ecx

mov esi,eax

mov eax,esi

mov \_a,eax

mov eax,\_d

sub eax,4

mov ebx,eax

mov eax,ebx

mov \_d,eax

jmp @l8

@l5:

mov eax,\_d

add eax,5

mov ebx,eax

mov eax,ebx

mov \_d,eax

@l8:

jmp @l0

@l1:

invoke crt\_printf,SADD("%d"),\_c

add esp,100h

pop ebp

invoke ExitProcess,NULL

end start

## 执行结果

正确

## 测试样例六（错误）

## 测试样例

const int a = 1;

int c;

void main(){

c=1;

printf c;

}

## 程序输出

出错，printf缺少括号

## 执行结果

错误

## 测试样例七（错误）

## 测试样例

void main()

{

int a;

scanf(a);

a = a+1;

printf("a=", a);

return

}

## 程序输出

出错，return缺少分号

## 执行结果

错误

## 测试样例八（错误）

## 测试样例

int g(int aa)

{

int a;

if(aa==1)

a=1;

else

a=aa\*g(aa-1);

}

void main()

{

int i;

scanf(i);

a=g(i);

printf(a);

}

## 程序输出

出错，标识符a未定义

## 执行结果

错误

## 测试样例九（错误）

## 测试样例

int a,b;

void function1(){

const int a=1,b=1;

printf("function1: ");

printf(a);

printf(b);

function2();

}

void function2(){

const int a=3,b=4;

printf("function2: ");

printf(a);

printf(b);

}

void main(){

int c[3];

b=a;

a=5;

c[0]=0;

c[1]=1;

c[2]=2;

a=c[0]+5;

function1();

printf("main: ");

printf(a);

printf(b);

function1();

printf(">");

}

## 程序输出

函数调用在定义之前

## 执行结果

错误

## 测试样例十（错误）

## 测试样例

int f(int a)

{

if(a==1)

a=1;

else

a=a\*f(a-1,1);

return(a);

}

void main()

{

int i,a;

printf("please input n =:\n");

scanf(i);

a=f(i);

printf("n! = %d\n",a);

}

## 程序输出

出错，函数调用时参数个数不符定义

## 执行结果

错误

1. **总结感想**

这次作业是上大学以来用时最长的一次程序设计，从开始到最后大概有几百小时。功能基本上完善，也进行了一些的优化，基本上达到了要求。

这次采用了C++以及面向对象的思想设计了整个编译器，但是由于时间紧迫，对于public和private成分没有考虑太清楚。总的来说，这个编译器可以产生符合386指令集规范的汇编代码，并集成Masm32可以在编译成功后直接运行32位应用程序。

我本来对C++不太熟练，这次选用C++就是为了借此机会熟悉一下C++，来夯实自己的基础。所以，在最开始的时候，还是遇到了不小的挫折，但是通过回顾史晓华老师在大二开设C++课和网上查阅资料，也逐渐适应了C++的风格。虽然程序在有些地方因为时间原因没来得及改还是比较非主流，但是我对C++的熟练程度却更深了一层。另外，因为这次大作业跟数据库课程设计同时进行，对比C++和C#，还是觉得C#编写面向对象的程序比较方便。

考虑到不同版本的Masm的功能和使用方法的差异以及手工使用Masm进行汇编连接很不方便，本系统自带了汇编器ml.exe和连接器link.exe，并在程序内集成，编译无错时直接汇编并连接生成目标程序，可调用system运行程序。

当然，整个编写过程中最难也是最有意思的地方在于优化，优化的方法繁多而复杂，是一个十分值得探讨的问题。关于对数组操作建立DAG图并导出代码我一直没有做好，这也是以后要思考的地方。优化的理论基础和实际应用还是有很大差别，锻炼了我联系理论的实践能力。

总之，这学期的编译技术课程设计让我收获很大。我不仅在这此设计上融合了包括高级语言程序设计、面向对象思想、软件工程等知识，更锻炼了实际的代码开发能力，对程序运行过程和计算机系统架构也有更深的认识。为我在计算机领域的工作打下了厚实的基础。

1. **申优文章**

本编译器采用了一遍扫描方法，采用面向对象的思想进行构建，各个主要部分抽象成类。主要的类有：词法分析器Lexer，语法分析器Parser，符号表管理器SymbolMgr，四元式管理器QuadrupleMgr，代码生成器CodeGenerator，数据流分析器DSAOptimizer，全剧寄存器分配器GRDOptimizer，局部公共表达式删除器CSDOptimizer。另外还有一些作为数据结构的类，主要有词语Token，符号表元素Item，四元式Quadruple，基本块BasicBlock，基本块集合BBSetMgr，DAG节点DAGNode，DAG表DAGTable等。为了方便数据在各个对象间传递并进行修改，同时减小时间空间的开销，本程序对所有的对象均用指针构造。

本程序错误处理综合在词法分析和语法分析中。在词法分析上，采用空格分割和词法相结合的方法划分词块和初步进行错误勘测。在语法分析方面，C0文法没有左递归但是有回溯问题。本程序采用预读字符发进行处理。如果改写文法消除回溯，是可以实现的，但是却将原文法中连贯语义的短语打乱了，给语义分析造成了一定的困难。所以我采用了预读字符的方法，在确定了产生式之后立即进行语义分析，并产生相应的四元式和进行错误处理。

本程序在符号表的组织上，没有采用栈式符号表，而是无序符号表，并用大表小表结合的方法，将函数内部的符号加入到小表中，从而让整个符号表结构更加清晰。同时，符号表中元素全部采用Item指针，同时Item的成员变量有指向父母表和子表的指针，从而保证了这种结构的有效性。

在中间代码方面，由于考虑到为使涉及四则运算的代码更易实现，选定了符号表项Item作为四元式的操作数，对于其它类型的如字符串、标签、临时变量等不是符号表的成员，我利用QuadrupleMgr新建临时的Item变量，用kind记录类型，从而保证了直接利用符号表项建立四元式的有效性。

本程序对于数组处理引入了IDX取址运算和IDXASN取址赋值运算。IDX第一个操作数是数组下标，第二个操作数是数组名，目的操作数是临时变量。IDXASN第一个操作数是值，第二个操作数是数组下标，目的操作数是数组名。同时，定义这样的运算符也为后面的DAG图的建立提供的方便。

最后是优化部分，本系统实现了局部公共子表达式删除优化（CSDOptimizer），全局寄存器分配优化（GRDOptimizer）等。其中全局寄存器分配优化采用了图着色算法，并使用数据流分析（DSAOptimizer）的活跃变量分析来构建冲突图。由于局部公共子表达式删除优化CSDOptimizer需要在基本块内进行，全局寄存器分配优化GRDOptimizer也需要函数块（过程块）的信息以及各基本块的冲突图信息。因此数据流分析GDAOptimizer需要最先执行。GDAOptimize分函数将基本块划成若干集合，并分别进行分析。

全局寄存器分配优化GRDOptimizer，跨越基本块，在函数（过程）范围内的，因此需要上一步GDAOptimizer生成的基本块集合，并且利用集合构建了冲突图ConflictTable，一个冲突图表ConflictTable是一个二维数组，行和列相同，都为项Item，当表格中值为1表示行与列有冲突。生成冲突图后实现图着色算法，为局部变量和参数变量分配全局寄存器，优化后的结果会直接反映到四元式管理器类QuadrupleMgr中的四元式数组。

最后是局部公共子表达式删除优化CSDOptimizer，它是在基本块基础上建立DAG图,，对于数组，正如前面所说，用IDX和IDXASN作为运算符针对数组构建节点。此外，对于数组不同的IDXASN操作，即使数组名节点号改变，原来的节点仍需要计算，除非数组名和左子节点都相同。CSDOptimizer每完成一个基本块的优化，都会把结果反映到四元式管理器类QuadrupleMgr中的四元式数组中。