**课 程 设 计 报 告**

设计题目：类C语言到汇编生成

班 级： 计算机1904

组长学号：20194687

组长姓名：伍孝飞

主 笔：伍孝飞 熊程林 唐知宸

设计时间：2021年7月

设计分工

组长学号及姓名：伍孝飞 20194687

分工：3

组员1学号及姓名： 唐知宸 20194623

分工：3

组员2学号及姓名：熊程林 20194604

分工：3

**摘 要**

本项目全程自己创建修改文法，定义四元式，四元式到目标代码生成。最终能够得到汇编代码在8086编译器上成功运行得到正确结果。

支持语法如下：

类型： int , char , bool

句子： 赋值语句，声明语句,循环语句（for,while）， 条件语句，函数，数组。

较复杂语句：函数递归，函数带参，临时变量声明，作用域限定，变量未定义检测，多重嵌套循环 ，逻辑表达式，数组下标嵌套表达式，循环语句声明临时变量，复杂复合语句。

前端文法采用递归下降子程序，前端词法生成函数表，函数参数表进行识别。

语义分析，使用前端词法进行词法结合，得到本小组自己规定的四元式，并且稍作优化，进行分块。

目标代码生成：最终目标是生成可直接运行的8086汇编语言代码。以链表形式读取文件中的四元式，根据分块生成活跃信息表，并对临时变量、函数参数的声明及其所有引用进行标注；遍历链表，形成汇编代码数据段；形成汇编代码堆栈段，以应对递归函数等堆栈需求；接着形成代码段，将链表中所含有的函数段提取出来，接着根据四元式优先对其进行目标代码生成工作；对剩余数据进行目标代码生成工作，形成最后的汇编代码。

**关键词**：编译原理，前端，后端，语法，四元式生成，目标代码生成

## 概述

整个完整的编译器包括编译器前端和编译器后端。编译器前端，包含词法分 析、语法分析和语义分析，是一个编译器的基础。

我们做的编译器前端中定义的 类C语言文法包括：变量和常量的声明，多维数组， 过程函数和递归函数的声明和调用，算术运算，逻辑运算，赋值语句，条件选择语句，循环语句。

编译器后端则包括了目标代码的生成和优化，真正把类C语言翻译成了单寄存器下的汇编语言。 我们完成的编译器实现了对基本C语言程序的编译、中间代码生成 和目标代码的生成。

程序编译过程中，作为输入的源程序首先经过词法分析器， 做相应判断和输出，生成 Token 序列；然后进入语法分析器，做相应判断和输出；接着，进入语义分析器，并将类C语言代码翻译成四元式，输出中间代码，并且分块。

在此过程中数据的越界和溢出检查，值单元存储分配信息，子函数和过程函数的参数传递与校验；紧接着四元式中间代码进入后端的优化模块，对中间代码进行优化，常数表达式节省（常数合并），公共子表达式节省（删除多 余运算），删除无用赋值；最后进入目标代码生成模块，生成多寄存器下的目标代码。

目标代码生成，对于给予的四元式进行连接生成汇编语言，能够在8086编译器成功运行。

## 课程设计任务

**2.1 设计任务**

编译原理课程兼有很强的理论性和实践性，是计算机专业的一门非常重要的专业基础课程，在系统软件中占有十分重要的地位。编译原理课程设计是本课程重要的综合实践教学环节，是对平时实验的一个补充。通过编译器相关子系统的设计，使学生能够更好地掌握编译原理的基本理论和编译程序构造的基本方法和技巧，融会贯通本课程所学专业理论知识；培养学生独立分析问题、解决问题的能力，以及系统软件设计的能力；培养学生的创新能力及团队协作精神。

**我们小组选的任务是：一个简单文法的编译器的设计与实现：**

定义一个类C语言文法（包含部分python文法），包括：

变量声明及定义语句；

常量声明语句；

算术运算表达式；

赋值语句；

数组的声明、赋值和调用；

If 语句；

While 语句；

For语句；  
 允许嵌套与递归的可含参函数的声明与调用；

**实现功能有：**

扫描器的设计实现；

语法分析器的设计实现；

词法分析器的设计实现；

前端报错的设计实现；

函数表的设计和实现；

中间代码生成器设计实现；

中间代码的简单优化；

寄存器分配；

目标代码生成器设计实现。

**2.2 设计要求**

1、在深入理解编译原理基本原理的基础上，对于选定的题目，以小组为单位，先确定设计方案；

2、设计系统的数据结构和程序结构，设计每个模块的处理流程。设计合理；

3、编程序实现系统，要求实现可视化的运行界面，界面应清楚地反映出系统的运行结果；

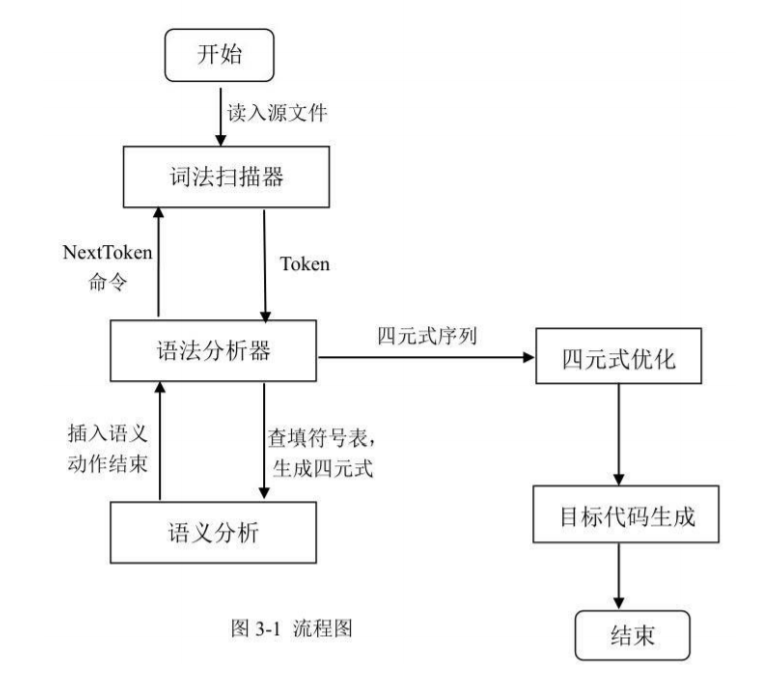
4、确定测试方案，选择测试用例，对系统进行测试；

5、运行系统并要通过验收，讲解运行结果，说明系统的特色和创新之处，并回答指导教师的提问；

1. 提交课程设计报告。

## 算法与数据结构

语法制导的编译器流程图：



**（1）文法**

文法部分包含语句定义，赋值语句，句子，复合句子，循环语句，主函数，函数，调用函数，函数声明，变量声明，条件语句，正值，数字，表达式。

文法高亮部分，为文法主体。

＜加法运算符＞ ::= +｜-  
＜乘法运算符＞  ::= \*｜/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞   ::= ０｜＜非零数字＞  
＜非零数字＞  ::= １｜．．．｜９  
＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

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

＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞

＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}

                  | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝| 0  
＜整数＞        ::= ［＋｜－］＜无符号整数＞

＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞

＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']' )}

                 //＜无符号整数＞表示数组元素的个数，其值需大于0

＜类型标识符＞      ::=  int | char

＜有返回值函数定义＞  ::=  ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'  
＜无返回值函数定义＞  ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜参数表＞    ::=  ＜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＞}   //[+|-]只作用于第一个<项>  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
＜因子＞    ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞           
＜语句＞    ::= ＜条件语句＞｜＜循环语句＞| '{'＜语句列＞'}'| ＜有返回值函数调用语句＞;   
                           |＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜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＞ //整型表达式之间才能进行关系运算

       ｜＜表达式＞    //表达式为整型，其值为0条件为假，值不为0时条件为真

＜循环语句＞   ::=  while '('＜条件＞')'＜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）词法分析**

词法分析部分包含识别单词和生成 Token 两部分，首先读入采用读文件的方式读入源文件，开始调用识别器，并根据相关信息生成 Token。最终，源程序经过词法分析器后生成了 Token 序列，便于以后的语法分析。

（**2）语法语义分析**

该模块包含语法分析和语义分析，语法分析部分使用递归下降子程序文法， 语义分析即是在语法分析过程中插入语义动作，生成四元式。利用函数信息表和符号表进行定义和重定义检查，类型匹配校验，函数参数传递，值单元存储分配信息，子函数和过程函数的参数传递与校验；程序执行前要先定义语义分析栈，和四元式队列，以便当语义动作到来时能直接使 用这些数据结构。

## 前端代码测试

当前语法分析已经大致完成，能够做到的目标以下示例(不完全包括，只有典型样例)

1：

int i , j , k

缺少 ;

2：

int i , j , k ,

缺少标识符

3:

sum = sum + ;

表达式错误

4:

int main(

主函数错误

5:

int i , j ;

k = 1;

存在未定义

6:

return n + f(temp1;

函数缺少 )

7:

int temp1 = n - 1

缺少 ;

8:

if(sum1 > sum2 )

{

flag = 1;

{}不匹配

9:

int fic(int n)

{

if(n == 1) return 1;

int temp2 = n - 1;

return n \*fi(temp2);

}

存在未定义

10:

int f(int n )//此处为函数声明

.....

int sum1 = f(10,1); //此处为函数调用

函数参数数目错误

11:

if(sum1 sum2 )

算术表达式错误

12:

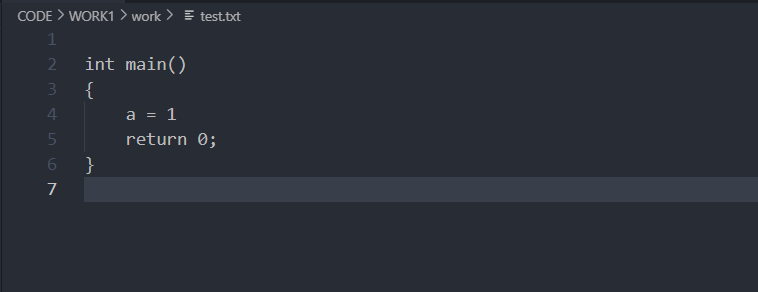
int f int n )//此处应为 函数声明

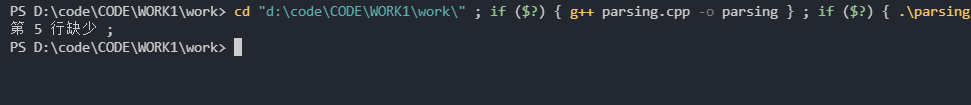
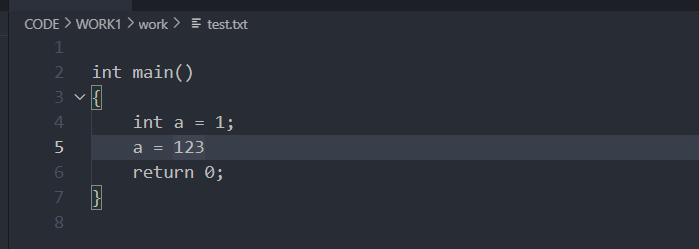
缺少 ;

13:

int f(int n //此处为 函数声明

缺少 )





**（3）四元式生成**

四元式生成：最终目标是生成完整的并且完成分块的四元式，输出到目标txt文件中，能够被读取并且生成目标代码。  
 首先定义各种四元式，包括函数，数组，for循环等，不同的四元式为不同状态，以压栈的形式形成一个状态栈，形成一个自动机将token串逐一读取并且识别当前状态，出现一个新的状态则将新的状态压入状态栈中，直到遇到该状态的结束符，则弹栈，改变当前状态，在某一状态下则执行对应状态的解析函数。例如，声明状态，赋值状态，for状态，函数状态等等，优先处理当前状态，已完成对多层次结构的四元式生成。  
 同时，用表达式函数，统一处理所有的表达式部分，完成&&，||，>=，（，）等20多个运算符的复杂表达式解析（首先通过递归，逐步去括号，再按照算符优先级解析生成四元式），同时表达式可以包含数组，函数调用的运算。

**（4）目标代码生成**

目标代码生成：最终目标是生成可直接运行的8086汇编语言代码。  
以链表形式读取文件中的四元式，根据分块生成活跃信息表，并对临时变量、函数参数的声明及其所有引用进行标注；  
遍历链表，形成汇编代码数据段；  
形成汇编代码堆栈段，以应对递归函数等堆栈需求；  
接着形成代码段，将链表中所含有的函数段提取出来，接着根据四元式优先对其进行目标代码生成工作；  
对剩余数据进行目标代码生成工作，形成最后的汇编代码。

## 编译器前端设计

### 4.1 扫描器

扫描器采用多数据读取，这样的操作好处是能够读取后面的数据，超前读取字符得到TOKEN串，能够更加准确的识别单词。

### 4.2 词法分析

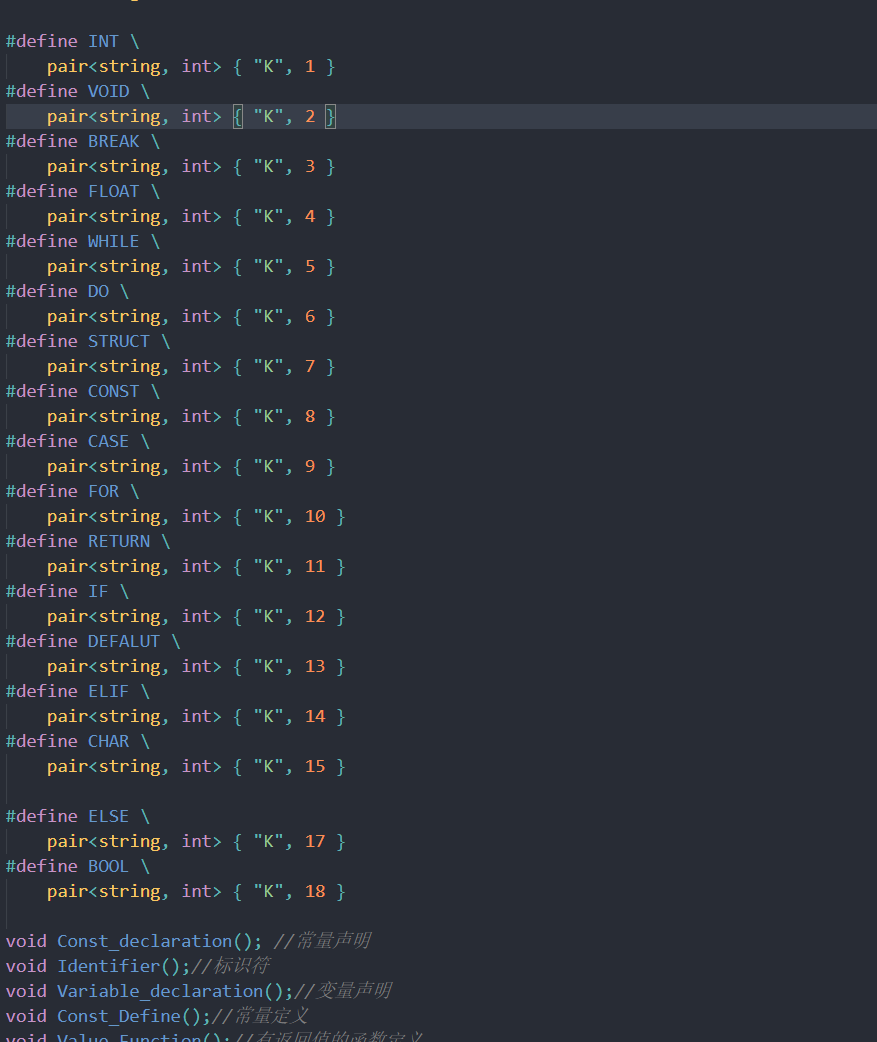
使用标准文法进行递归子程序进行识别，自己构建了函数表和函数参数表，使用的map进行log(N)查询。

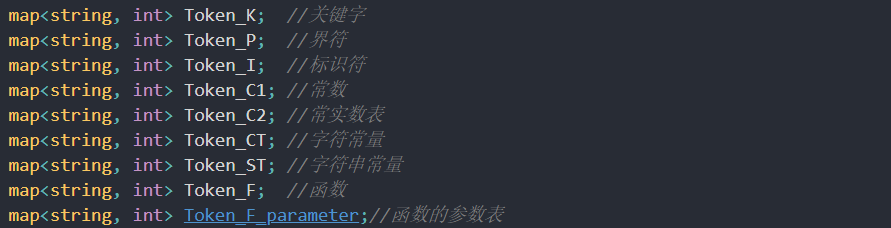
### 算法设计

由于词法程序已经能够实现了，我们就采取对于词法的成分进行识别，进行成分到文法的转换递归，再添加表达式的优化和模糊处理，就能够得到性能较好的前端设计。

若一个非终结符有多个拥有公共前缀的候选式，对该非终结符进行推导的时候就难以选择合适的产生式进行推导，因此会产生回溯问题。这里，我们便可以使用提取左因子的方式在一定程度上规避回溯问题。

利用词法和语法，能够更好的兼容和修改框架和结构。





**5 编译器后端设计**

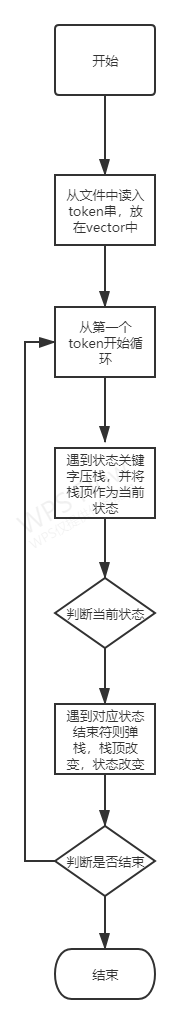
**5.1 四元式生成**

**5.1.1 四元式生成算法设计**

首先定义各种四元式，包括函数，数组，for循环等，不同的四元式为不同状态，以压栈的形式形成一个状态栈，形成一个自动机将token串逐一读取并且识别当前状态，出现一个新的状态则将新的状态压入状态栈中，直到遇到该状态的结束符，则弹栈，改变当前状态，在某一状态下则执行对应状态的解析函数。例如，声明状态，赋值状态，for状态，函数状态等等，优先处理当前状态，已完成对多层次结构的四元式生成。  
同时，用表达式函数，统一处理所有的表达式部分，完成&&，||，>=，（，）等20多个运算符的复杂表达式解析（首先通过递归，逐步去括号，再按照算符优先级解析生成四元式），同时表达式可以包含数组，函数调用的运算。

**5.1.2 四元式生成流程图**

流程图如下：



**5.1.3 代码实现（表达式处理函数）**

string expressionStack(ofstream& out,vector<string> data)//输入一个复杂表达式，通过压栈弹栈的方式生成四元式

{

//printdata(data);

if (data.size() == 3 && (isCT || isconst || isST))//当栈顶为等号时，直接赋值，直接处理，例如a=b或a='b'

{

string ct;

if (isCT) ct = "\'" + data[2] + "\'", isCT = 0;//加上引号

else if(isST) ct = "\"" + data[2] + "\"", isST = 0;//加上双引号

else ct = data[2];

outfile(out,HS.top(), ct, "\_", data[0]);

return "end";//直接结束

}

stack< string > s;//负责把栈反过来的数据正向弹出来

vector<string> t;//负责把没有括号的表达式即简单表达式保存起来

if (!haveParentheses(data))//判断是否有括号，没有括号直接处理

{

//~取反 ,单目运算符

for (unsigned int i = 0; i < data.size(); i++)//循环的目的：找到 ~ 的位置

{

if (data[i] == "++" || data[i] == "--" || data[i] == "!" || data[i] == "~" || (i==0 && (data[i] == "+" || data[i] == "-")))

{

num\_t++;//t的数量加一

outfile(out,data[i], "0", data[i + 1], intos());//写入一个四元组

data[i] = intos();//将表达式更改为新表达式，将a\*b->t

data.erase(data.begin() + i);//删除一个多余的元素

string ret = expressionStack(out,data);//将新表达式再放入函数中处理，直到整个表达式没有单目运算符时返回

return ret;//返回则直接跳出

}

}

//先解决\* / %运算，再解决+ -运算，按照符号优先级

for (unsigned int i = 0; i < data.size(); i++)//循环的目的：找到\* / %的位置，分成两个循环的目的：先处理完\* / %

{

if (data[i] == "\*" || data[i] == "/" || data[i] == "%")

{

num\_t++;//t的数量加一

outfile(out,data[i], data[i - 1], data[i + 1], intos());//写入一个四元组

data[i - 1] = intos();//将表达式更改为新表达式，将a\*b->t

data.erase(data.begin() + i, data.begin() + i + 2);//删除两个多余的元素

string ret = expressionStack(out,data);//将新表达式再放入函数中处理，直到整个表达式没有\* / %时返回

return ret;//返回则直接跳出

}

}

//处理完优先级高的符号后处理+ -

for (unsigned int i = 0; i < data.size(); i++)

{

if (data[i] == "+" || data[i] == "-")//循环找到第一个+，-

{

num\_t++;

outfile(out,data[i], data[i - 1], data[i + 1], intos());

data[i - 1] = intos();

data.erase(data.begin() + i, data.begin() + i + 2);//删除两个多余元素，a+b->t

string ret = expressionStack(out,data);//把新的表达式递归继续处理，直到没有+ -

return ret;//变成t后跳出

}

}

//>> <<

for (unsigned int i = 0; i < data.size(); i++)

{

if (data[i] == ">>" || data[i] == "<<")

{

num\_t++;//t的数量加一

outfile(out,data[i], data[i - 1], data[i + 1], intos());//写入一个四元组

data[i - 1] = intos();//将表达式更改为新表达式，将a>>b->t

data.erase(data.begin() + i, data.begin() + i + 2);//删除两个多余的元素

string ret = expressionStack(out,data);//将新表达式再放入函数中处理，直到整个表达式没有 << >> 时返回

return ret;//返回则直接跳出

}

}

//= == > < >= <= !=

for (unsigned int i = 0; i < data.size(); i++)

{

if (data[i] == "=" || data[i] == "==" || data[i] == "<" || data[i] == ">" || data[i] == "<=" || data[i] == ">=" || data[i] == "!=")

{

if (data[i] == "=") outfile(out,data[i], data[i + 1], "\_", data[i - 1]);//写入一个四元组

else num\_t++,outfile(out,data[i], data[i - 1], data[i + 1], intos());//写入一个四元组,//用到intos（），t的数量加一

data[i - 1] = intos();//将表达式更改为新表达式，将a\*b->t

data.erase(data.begin() + i, data.begin() + i + 2);//删除两个多余的元素

string ret = expressionStack(out,data);//将新表达式再放入函数中处理，直到整个表达式没有上述符号时返回

return ret;//返回则直接跳出

}

}

//&& ||

for (unsigned int i = 0; i < data.size(); i++)

{

if (data[i] == "&&" || data[i] == "||")

{

num\_t++;//t的数量加一

outfile(out,data[i], data[i - 1], data[i + 1], intos());//写入一个四元组

data[i - 1] = intos();//将表达式更改为新表达式，将a||b->t

data.erase(data.begin() + i, data.begin() + i + 2);//删除两个多余的元素

string ret = expressionStack(out,data);//将新表达式再放入函数中处理，直到整个表达式没有 && || 时返回

return ret;//返回则直接跳出

}

}

}

else

{

//如果有括号，先去括号

for (unsigned int i = 0; i <= data.size()+1; i++)

{

//cout << "!!!" << endl;

if (i == data.size()+1 || (i< data.size() && data[i] == ")"))//等于“）”时处理，或者循环到字符串尾部时处理

{

//cout << "!!!" << endl;

i++;//往后移一位，防止压栈压入 “）”

t.clear();//清空t组

while (!EXS.empty() && EXS.top() != "(")//弹栈，弹到“（”

{

t.push\_back(EXS.top());

EXS.pop();

}

if(!EXS.empty()) EXS.pop();//弹出一个"("括号

reverse(t.begin(), t.end());//倒置

//printdata(t);

string ret = expressionStack(out,t);//弹出部分的简单表达式递归处理

//cout << "??????" << ret << " " << i << " " << data.size() << endl;

EXS.push(ret);//处理结束，将符号t\_n压栈

}

if (i < data.size()) EXS.push(data[i]);//不等于 “）” 则压栈,字符串结束不压栈

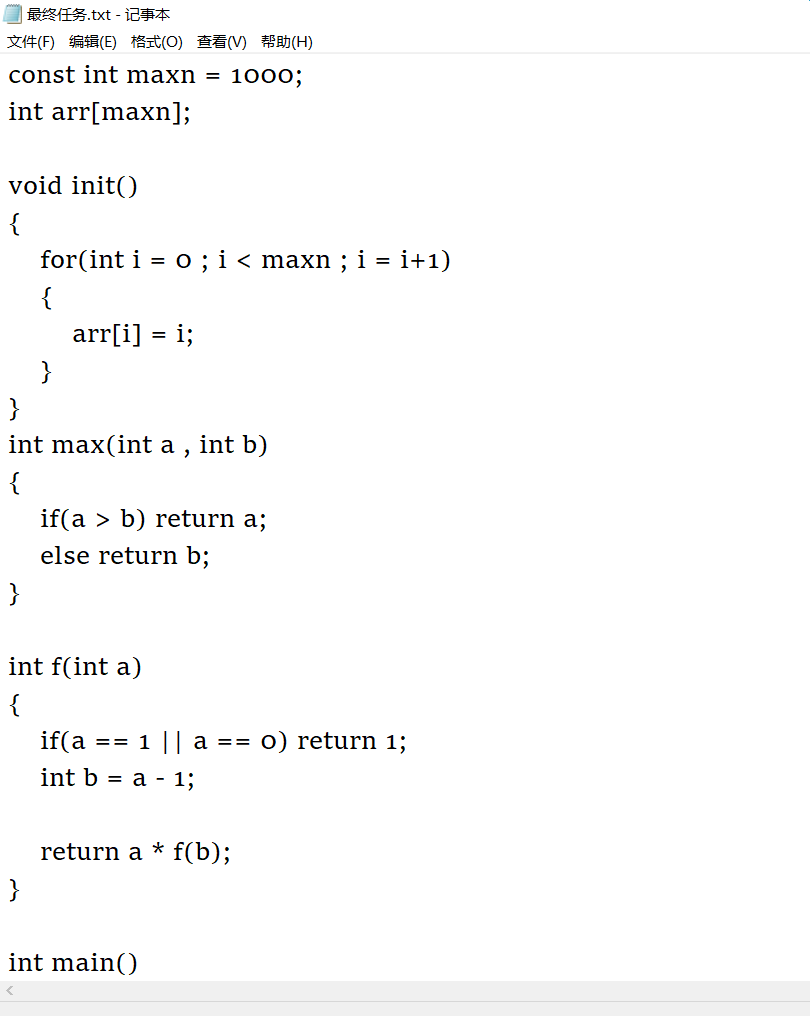
}

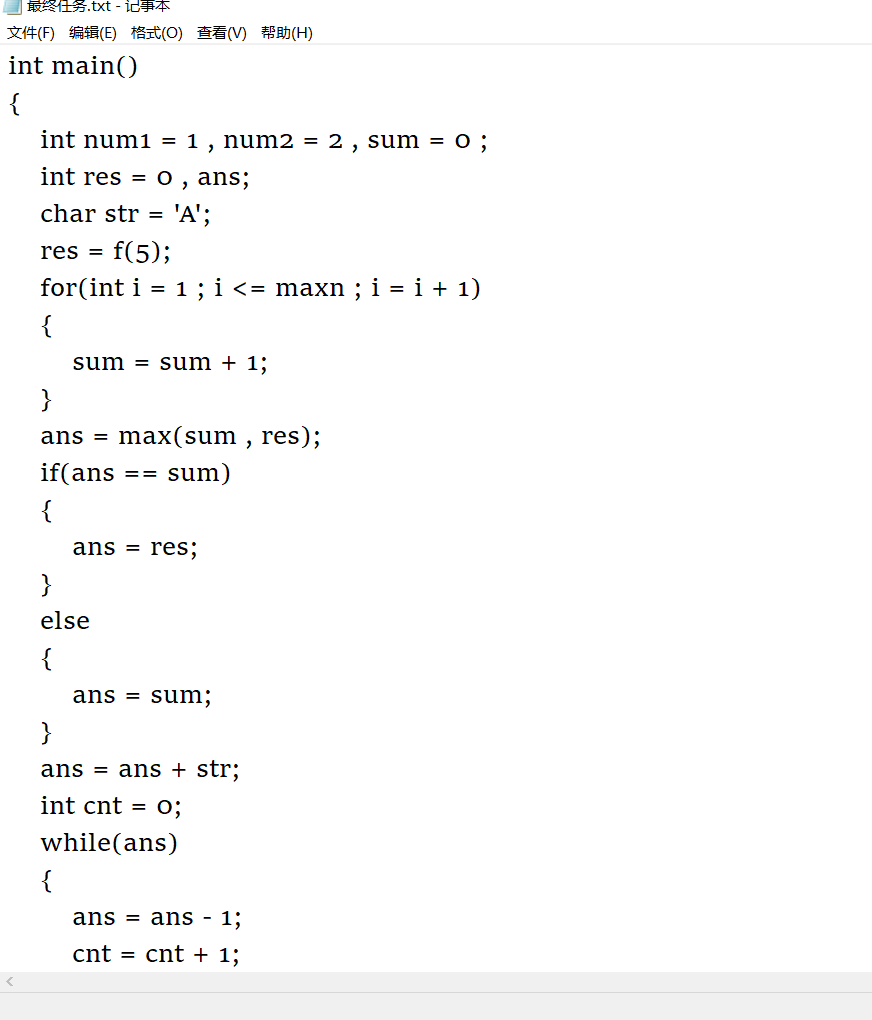
}

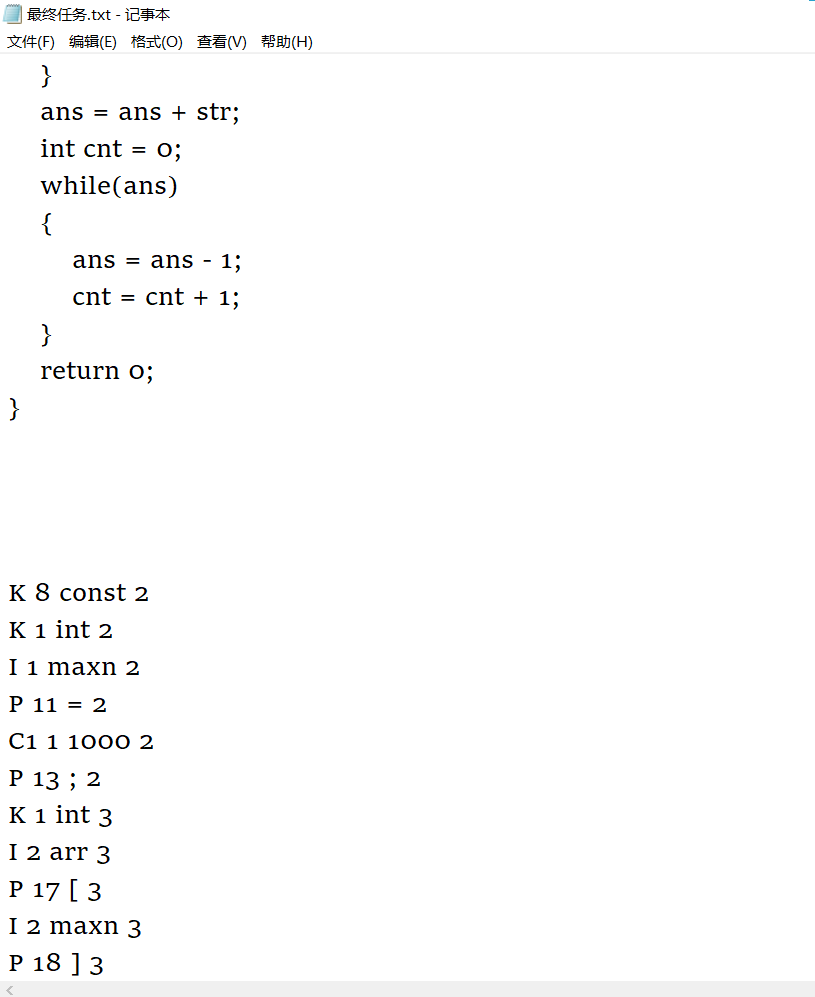
return intos();

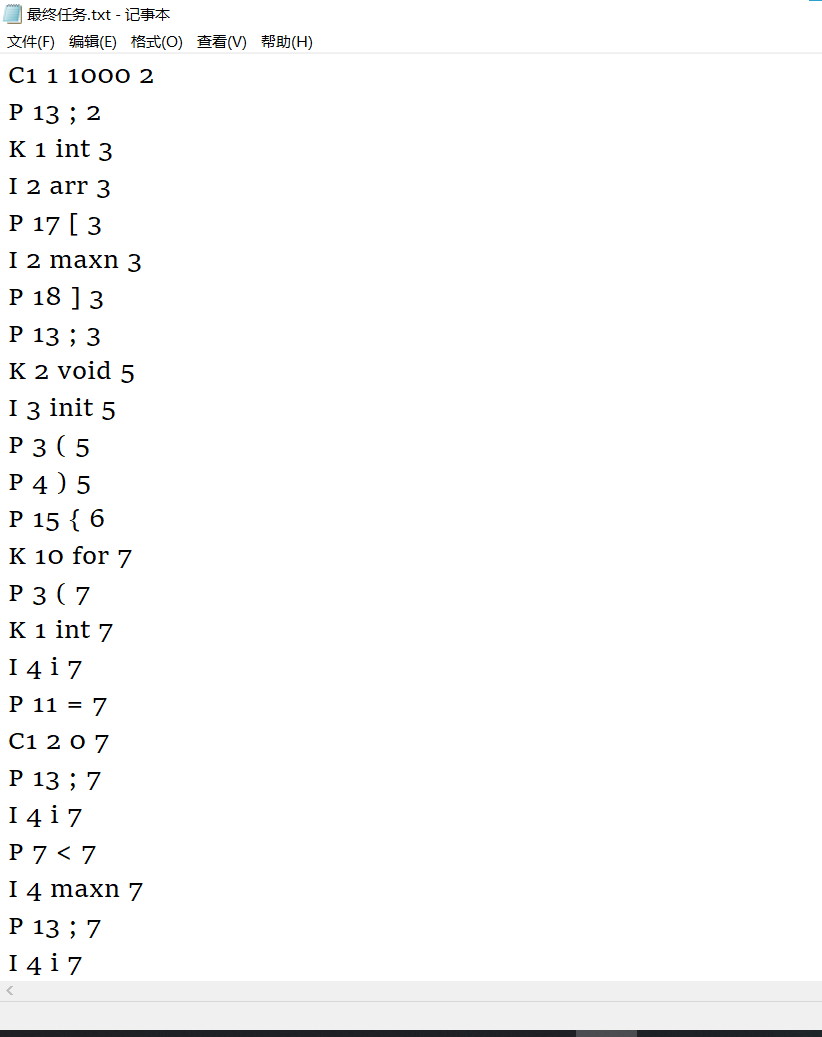
}

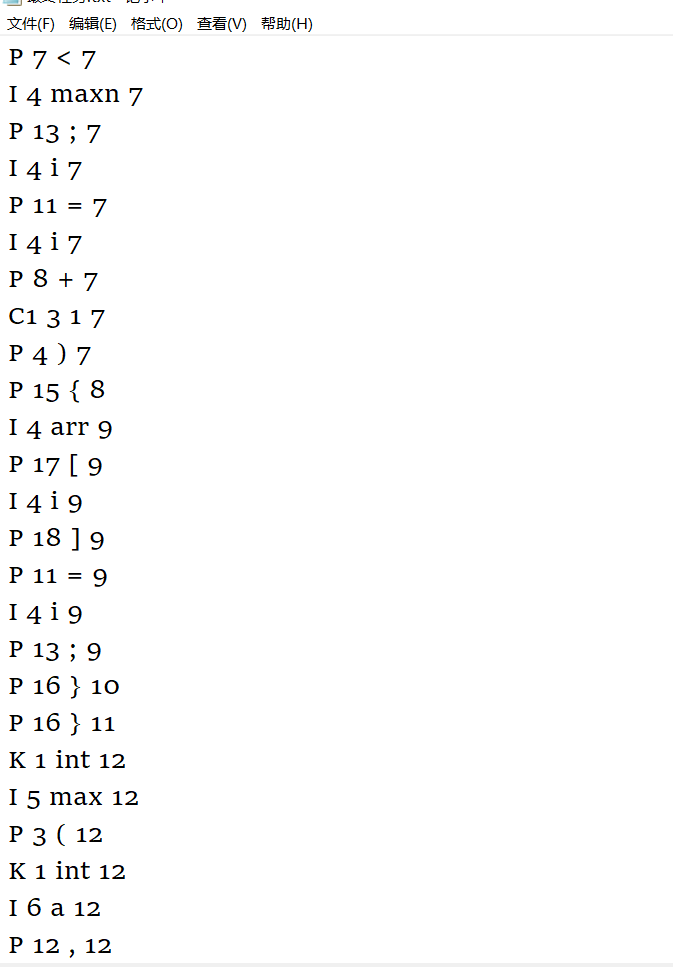
**5.1.4 测试样例以及结果**

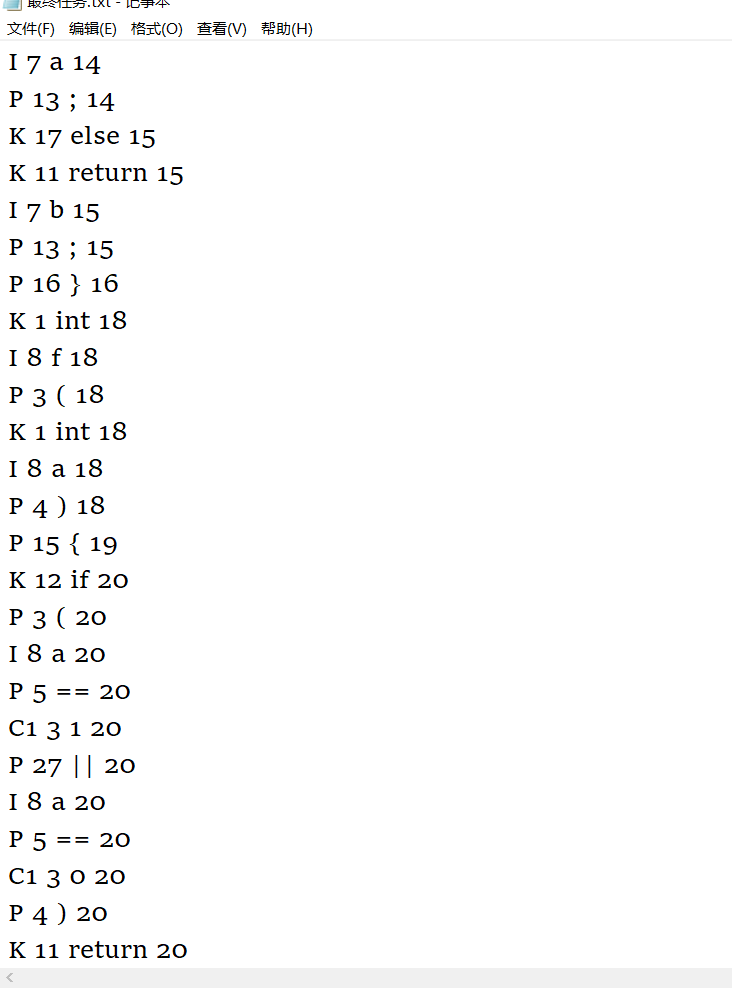


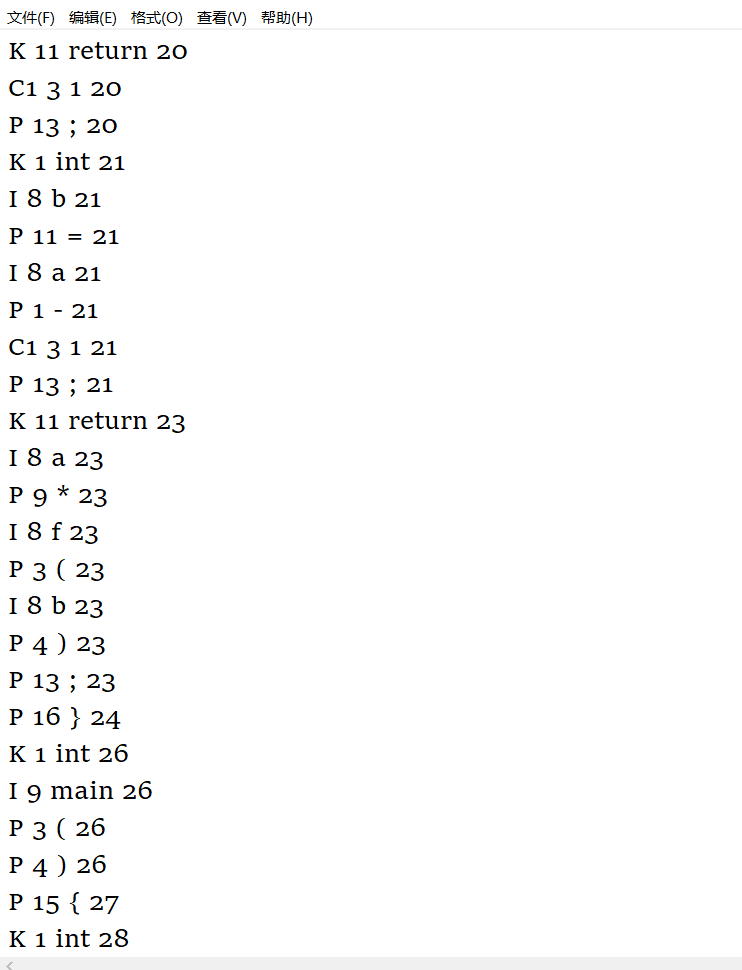


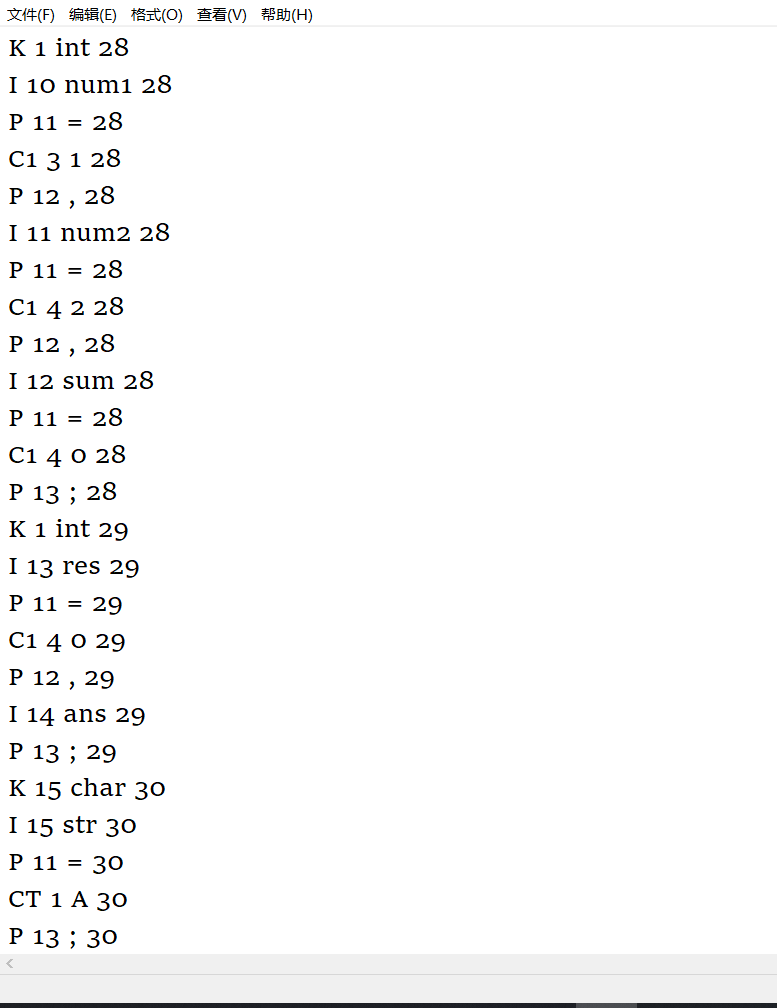


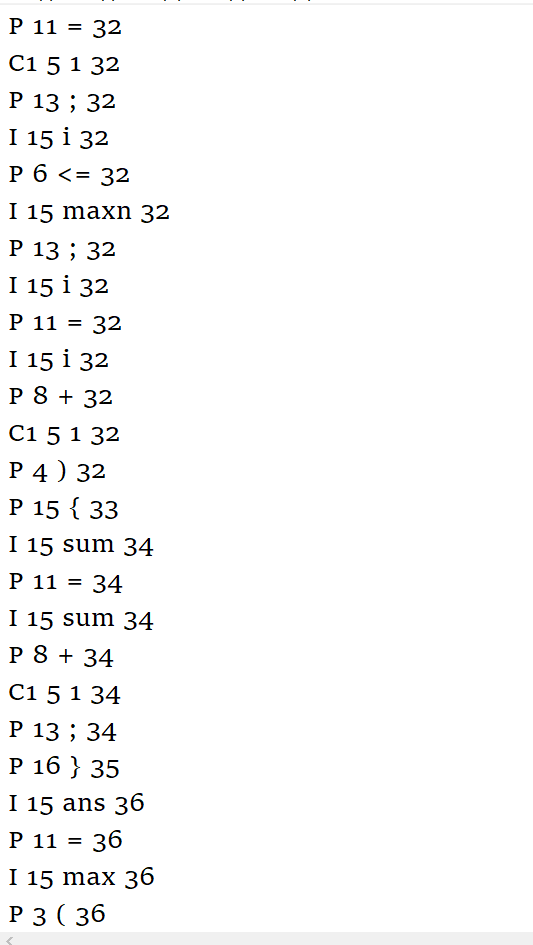


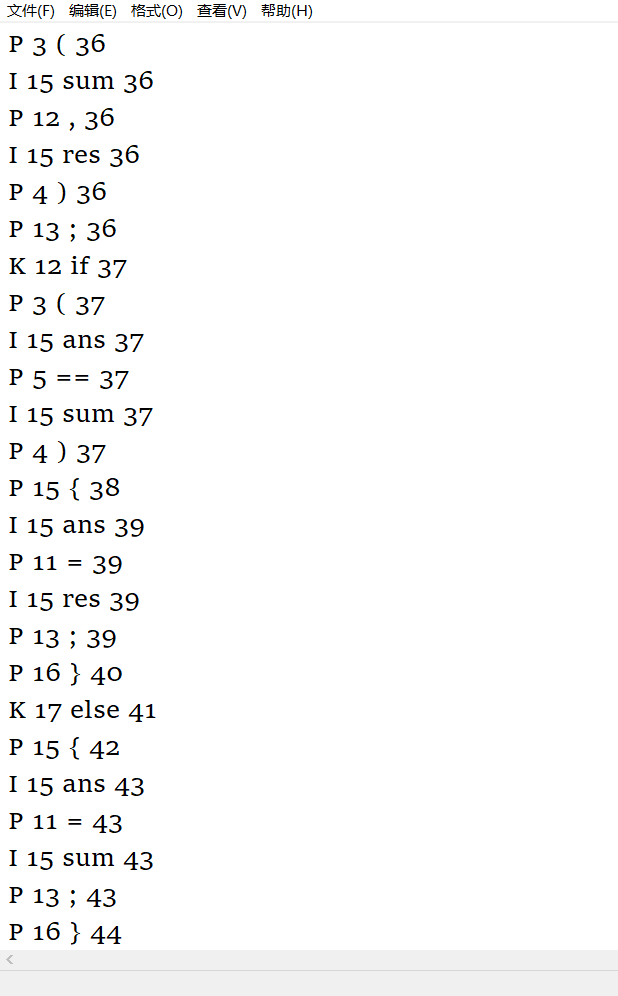


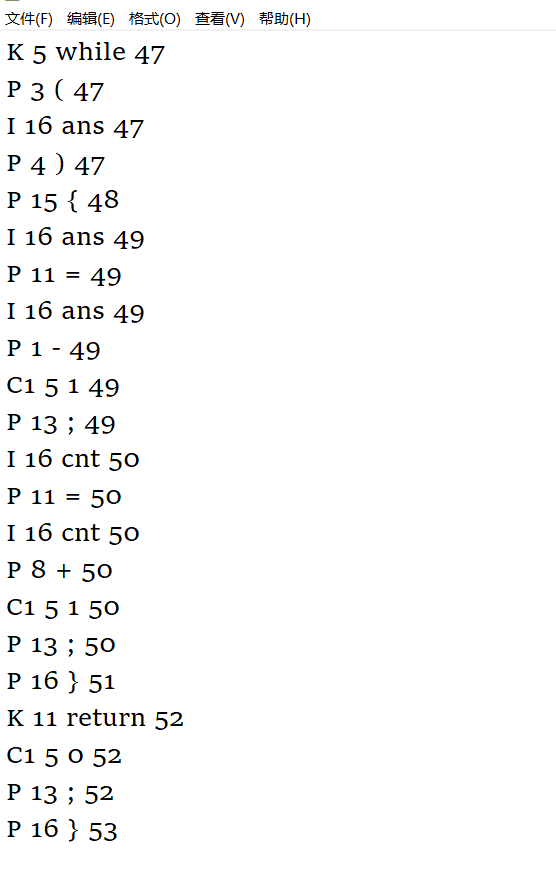












输出的四元式结果为：

int maxn \_ \_

= 1000 \_ maxn

int (arr|maxn) \_ \_

f init void \_

int i \_ \_

= 0 \_ i

for \_ \_ \_

< i maxn t0

do t0 \_ \_

+ i 1 t1

= t1 \_ i

= i \_ (arr|i)

fe \_ \_ \_

f end init \_

declarP max (int|int) (a|b)

f max int \_

> a b t2

if t2 \_ \_

return a \_ \_

ie 0 \_ \_

else \_ \_ \_

return b \_ \_

ie 2 \_ \_

f end max \_

declarP f (int) (a)

f f int \_

== a 1 t3

== a 0 t4

|| t3 t4 t5

if t5 \_ \_

return 1 \_ \_

ie 0 \_ \_

int b \_ \_

- a 1 t6

= t6 \_ b

call f \_ (b)

\* a f\_ret t7

return t7 \_ \_

f end f \_

f main int \_

int num1 \_ \_

= 1 \_ num1

int num2 \_ \_

= 2 \_ num2

int sum \_ \_

= 0 \_ sum

int res \_ \_

= 0 \_ res

int ans \_ \_

char str \_ \_

= 'A' \_ str

call f \_ (5)

= f\_ret \_ res

int i \_ \_

= 1 \_ i

for \_ \_ \_

<= i maxn t8

do t8 \_ \_

+ i 1 t9

= t9 \_ i

+ sum 1 t10

= t10 \_ sum

fe \_ \_ \_

call max \_ (sum|res)

= max\_ret \_ ans

== ans sum t11

if t11 \_ \_

= res \_ ans

ie 0 \_ \_

else \_ \_ \_

= sum \_ ans

ie 2 \_ \_

+ ans str t12

= t12 \_ ans

int cnt \_ \_

= 0 \_ cnt

while \_ \_ \_

!= ans 0 t13

do t13 \_ \_

- ans 1 t14

= t14 \_ ans

+ cnt 1 t15

= t15 \_ cnt

we \_ \_ \_

return 0 \_ \_

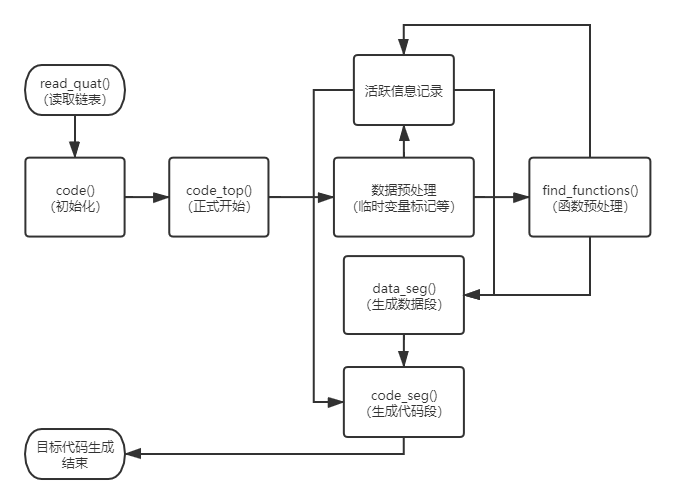
f end main \_

**5.2 目标代码生成**

**5.2.1 目标代码生成算法设计**

以链表形式读取文件中的四元式，根据分块生成活跃信息表，并对临时变量、函数参数的声明及其所有引用进行标注；遍历链表，形成汇编代码数据段；形成汇编代码堆栈段，以应对递归函数等堆栈需求；接着形成代码段，将链表中所含有的函数段提取出来，接着根据四元式优先对其进行目标代码生成工作；对剩余数据进行目标代码生成工作，形成最后的汇编代码。

**5.2.2 目标代码生成——顶层流程图**



**5.2.3 目标代码生成具体代码实现（部分）**

/\*数据预处理\*/

void code\_top(block\* head)

{

block\* cur\_block = head;

quat\* cur\_quat = cur\_block->q\_head;

quat\* temp\_quat = cur\_quat;

/\*去除声明语句,存进标识符数组\*/

int arr\_mark = 0;

int i = 0;

while (cur\_block != NULL)

{

cur\_quat = cur\_block->q\_head;

while (cur\_quat != NULL)

{

if (strcmp(cur\_quat->Q[0], "int") == 0 || strcmp(cur\_quat->Q[0], "char") == 0)

{

if (cur\_quat->Q[1][0] != '(') //不是数组

{

if (i > 0)

{

int j = 0;

for (j ; j < i; j++)

{

if (strcmp(biaoshifu[j], cur\_quat->Q[1]) == 0)

break;

}

if (strcmp(biaoshifu[j], cur\_quat->Q[1]) == 0)

{

int block\_mark = 1;

arr\_mark++;

char t1[5] = { 0 };

char t2[20] = { 0 };

itoa(arr\_mark, t1, 10);

strcpy(t2, cur\_quat->Q[1]);

strcat(t2, t1);

quat\* q1 = cur\_quat->next;

while (q1 != NULL)

{

if (strcmp(q1->Q[1], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[1], '\0', sizeof(q1->Q[1]));

strcpy(q1->Q[1], t2);

}

else if (strcmp(q1->Q[2], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[2], '\0', sizeof(q1->Q[2]));

strcpy(q1->Q[2], t2);

}

else if (strcmp(q1->Q[3], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[3], '\0', sizeof(q1->Q[3]));

strcpy(q1->Q[3], t2);

}

if (q1->Q[1][0]=='(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[1][j] != '|')

{

name[j - 1] = q1->Q[1][j];

j++;

}

j++;

int k = 0;

while (q1->Q[1][j] != ')')

{

num[k] = q1->Q[1][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[1], '\0', sizeof(q1->Q[1]));

memcpy(q1->Q[1], newch,30);

}

}

else if (q1->Q[2][0] == '(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[2][j] != '|')

{

name[j - 1] = q1->Q[2][j];

j++;

}

j++;

int k = 0;

while (q1->Q[2][j] != ')')

{

num[k] = q1->Q[2][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[2], '\0', sizeof(q1->Q[2]));

memcpy(q1->Q[2], newch,30);

}

}

else if (q1->Q[3][0] == '(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[3][j] != '|')

{

name[j - 1] = q1->Q[3][j];

j++;

}

j++;

int k = 0;

while (q1->Q[3][j] != ')')

{

num[k] = q1->Q[3][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[3], '\0', sizeof(q1->Q[3]));

memcpy(q1->Q[3], newch, 30);

}

}

q1 = q1->next;

}

block\* b1 = cur\_block->next;

if (b1 != NULL)

{

q1 = b1->q\_head;

while (b1 != NULL)

{

q1 = b1->q\_head;

while (q1 != NULL)

{

if (strcmp(q1->Q[1], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[1], '\0', sizeof(q1->Q[1]));

strcpy(q1->Q[1], t2);

}

else if (strcmp(q1->Q[2], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[2], '\0', sizeof(q1->Q[2]));

strcpy(q1->Q[2], t2);

}

else if (strcmp(q1->Q[3], cur\_quat->Q[1]) == 0)

{

memset(q1->Q[3], '\0', sizeof(q1->Q[3]));

strcpy(q1->Q[3], t2);

}

if (q1->Q[1][0] == '(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[1][j] != '|')

{

name[j - 1] = q1->Q[1][j];

j++;

}

j++;

int k = 0;

while (q1->Q[1][j] != ')')

{

num[k] = q1->Q[1][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[1], '\0', sizeof(q1->Q[1]));

memcpy(q1->Q[1], newch, 30);

}

}

else if (q1->Q[2][0] == '(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[2][j] != '|')

{

name[j - 1] = q1->Q[2][j];

j++;

}

j++;

int k = 0;

while (q1->Q[2][j] != ')')

{

num[k] = q1->Q[2][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[2], '\0', sizeof(q1->Q[2]));

memcpy(q1->Q[2], newch, 30);

}

}

else if (q1->Q[3][0] == '(')

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (q1->Q[3][j] != '|')

{

name[j - 1] = q1->Q[3][j];

j++;

}

j++;

int k = 0;

while (q1->Q[3][j] != ')')

{

num[k] = q1->Q[3][j];

k++;

j++;

}

if (strcmp(num, cur\_quat->Q[1]) == 0)

{

strcat(num, t1);

char newch[30] = { 0 };

strcat(newch, "(");

strcat(newch, name);

strcat(newch, "|");

strcat(newch, num);

strcat(newch, ")");

memset(q1->Q[3], '\0', sizeof(q1->Q[3]));

memcpy(q1->Q[3], newch, 30);

}

}

if (strcmp(q1->Q[0], "for") == 0 || strcmp(q1->Q[0], "while") == 0 || strcmp(q1->Q[0], "if") == 0)

{

block\_mark++;

}

else if (strcmp(q1->Q[0], "fe") == 0 || strcmp(q1->Q[0], "we") == 0 || strcmp(q1->Q[0], "ie") == 0 || (strcmp(q1->Q[0], "f") == 0 && strcmp(q1->Q[1], "end") == 0))

{

block\_mark--;

}

q1 = q1->next;

if (block\_mark == 0)

break;

}

if (q1 == NULL)

{

b1 = b1->next;

continue;

}

if (block\_mark == 0)

break;

}

}

memset(cur\_quat->Q[1], '\0', sizeof(cur\_quat->Q[1]));

strcpy(cur\_quat->Q[1], t2);

}

}

temp\_quat = cur\_quat;

strcpy(biaoshifu[i], cur\_quat->Q[1]);

cur\_quat = cur\_quat->next;

delete\_quat(cur\_block, temp\_quat);

i++;

}

else

{

char name[30] = { 0 };

char num[20] = { 0 };

int j = 1;

while (cur\_quat->Q[1][j] != '|')

{

name[j - 1] = cur\_quat->Q[1][j];

j++;

}

j++;

int k = 0;

while (cur\_quat->Q[1][j] != ')')

{

num[k] = cur\_quat->Q[1][j];

k++;

j++;

}

if (num[0] >= '0' && num[0] <= '9') //数组长度为定值

{

}

else //数组长度为变量

{

block\* temp\_b = head;

quat\* temp\_q = temp\_b->q\_head;

while (temp\_b != NULL)

{

temp\_q = temp\_b->q\_head;

while (temp\_q != NULL)

{

if (strcmp(temp\_q->Q[3], num) == 0)

{

break;

}

temp\_q = temp\_q->next;

}

if (temp\_q == NULL)

{

temp\_b = temp\_b->next;

continue;

}

if (strcmp(temp\_q->Q[3], num) == 0)

break;

}

memset(num, '\0', sizeof(num));

strcpy(num, temp\_q->Q[1]);

}

temp\_quat = cur\_quat;

strcat(num, name);

strcpy(biaoshifu[i], num);

cur\_quat = cur\_quat->next;

delete\_quat(cur\_block, temp\_quat);

i++;

}

}

else if (strcmp(cur\_quat->Q[0], "do") == 0)

{

temp\_quat = cur\_quat;

cur\_quat = cur\_quat->next;

delete\_quat(cur\_block, temp\_quat);

}

else

cur\_quat = cur\_quat->next;

}

if (cur\_block->next == NULL)

break;

else

cur\_block = cur\_block->next;

}

data\_seg(biaoshifu);

code\_seg(head);

}

/\*数据段生成\*/

void data\_seg(char biaoshifu[][20])

{

FILE\* fp;

fp = fopen("result.txt", "w");

fprintf(fp, "DSEG SEGMENT\n");

printf("DSEG SEGMENT\n");

int i = 0;

while (strcmp(biaoshifu[i], ""))

{

if (biaoshifu[i][0] >= '0' && biaoshifu[i][0] <= '9') //是数组

{

int j = 0;

char num[30] = { 0 };

while (biaoshifu[i][j] >= '0' && biaoshifu[i][j] <= '9')

{

num[j] = biaoshifu[i][j];

j++;

}

int k = 0;

char name[30] = { 0 };

while (biaoshifu[i][j] != '\0')

{

name[k] = biaoshifu[i][j];

j++;

k++;

}

fprintf(fp, "%s DW %s DUP(0)\n", name, num);

printf("%s DW %s DUP(0)\n", name, num);

i++;

}

else

{

fprintf(fp, "%s DW 0\n", biaoshifu[i]);

printf("%s DW 0\n", biaoshifu[i]);

i++;

}

}

fprintf(fp, "DSEG ENDS\n\n");

printf("DSEG ENDS\n\n");

fprintf(fp, "SSEG SEGMENT STACK\nSKTOP DW 300 DUP(0)\nSSEG ENDS\n\n");

printf("SSEG SEGMENT STACK\nSKTOP DW 300 DUP(0)\nSSEG ENDS\n\n");

fclose(fp);

}

/\*函数预处理\*/

void find\_functions()

{

block\* temp\_b1 = temp\_b;

quat\* temp\_q1 = temp\_b1->q\_head;

block\* temp\_b2 = temp\_b; //函数头

quat\* temp\_q2 = temp\_b1->q\_head;

block\* temp\_b3 = temp\_b; //函数尾

quat\* temp\_q3 = temp\_b1->q\_head;

char temp\_arr[5][20] = { 0 };

int k = 0;

while (temp\_b1 != NULL) //找到一个函数头

{

temp\_q1 = temp\_b1->q\_head;

while (temp\_q1 != NULL)

{

if (strcmp(temp\_q1->Q[0], "f") == 0 && strcmp(temp\_q1->Q[1], "end") != 0)

{

temp\_b2 = temp\_b1;

temp\_q2 = temp\_q1;

init\_quat(temp\_b1);

strcpy(temp\_b1->q\_tail->Q[0], "int"); //函数类型待定

strcpy(temp\_b1->q\_tail->Q[1], temp\_q1->Q[1]);

strcat(temp\_b1->q\_tail->Q[1], "\_ret");

strcpy(temp\_b1->q\_tail->Q[2], "\_");

strcpy(temp\_b1->q\_tail->Q[3], "\_"); //函数名\_ret为函数返回值

break;

}

else if (strcmp(temp\_q1->Q[0], "declarP") == 0)

{

temp\_b2 = temp\_b1;

temp\_q2 = temp\_q1;

init\_quat(temp\_b1);

strcpy(temp\_b1->q\_tail->Q[0], "int"); //函数类型待定

strcpy(temp\_b1->q\_tail->Q[1], temp\_q1->Q[1]);

strcat(temp\_b1->q\_tail->Q[1], "\_ret");

strcpy(temp\_b1->q\_tail->Q[2], "\_");

strcpy(temp\_b1->q\_tail->Q[3], "\_"); //函数名\_ret为函数返回值

int i = 1;

char c;

c = temp\_q1->Q[3][1];

while (c != ')')

{

int j = 0;

while (c != '|' && c != ')')

{

temp\_arr[k][j] = c;

i++;

j++;

c = temp\_q1->Q[3][i];

if (c == '|' || c == ')')

{

break;

}

}

init\_quat(temp\_b1);

strcpy(temp\_b1->q\_tail->Q[0], "int");

strcpy(temp\_b1->q\_tail->Q[1], temp\_arr[k]);

strcat(temp\_b1->q\_tail->Q[1], temp\_q1->Q[1]);

strcpy(temp\_b1->q\_tail->Q[2], "\_");

strcpy(temp\_b1->q\_tail->Q[3], "\_");

k++;

if (c == ')')

{

break;

}

i++;

c = temp\_q1->Q[3][i];

}

char temp[30] = { 0 };

temp[0] = '(';

strcat(temp, temp\_arr[0]);

strcat(temp, temp\_q1->Q[1]);

for (int n = 1; n < k; n++)

{

strcat(temp, "|");

strcat(temp, temp\_arr[n]);

strcat(temp, temp\_q1->Q[1]);

}

strcat(temp, ")");

strcpy(temp\_q1->Q[3], temp);

break;

}

temp\_q1 = temp\_q1->next;

}

if (temp\_q1 == NULL)

{

temp\_b1 = temp\_b1->next;

temp\_b = temp\_b1;

continue;

}

else if (strcmp(temp\_q1->Q[0], "f") == 0 && strcmp(temp\_q1->Q[1], "end") != 0)

break;

else if (strcmp(temp\_q1->Q[0], "declarP") == 0)

break;

}

temp\_b = temp\_b1;

if (temp\_b == NULL)

{

return;

}

temp\_b3 = temp\_b2->next;

temp\_q3 = temp\_b3->q\_head;;

while (temp\_b3 != NULL) //找到该函数结尾

{

temp\_q3 = temp\_b3->q\_head;

while (temp\_q3 != NULL)

{

if (strcmp(temp\_q3->Q[0], "f") == 0 && strcmp(temp\_q3->Q[1], "end") == 0)

{

break;

}

temp\_q3 = temp\_q3->next;

}

if (temp\_q3 == NULL)

{

temp\_b3 = temp\_b3->next;

temp\_b = temp\_b3;

continue;

}

else if (strcmp(temp\_q3->Q[0], "f") == 0 && strcmp(temp\_q3->Q[1], "end") == 0)

break;

}

while (temp\_q1 != temp\_q3 && temp\_b1 != temp\_b3->next)

{

if (temp\_q1 != temp\_q2)

temp\_q1 = temp\_b1->q\_head;

if (temp\_q1 == temp\_q3)

break;

while (temp\_q1 != NULL)

{

for (int i = 0; i < k; i++)

{

for (int j = 0; j < 4; j++)

{

if (strcmp(temp\_q1->Q[j], temp\_arr[i]) == 0)

{

strcat(temp\_q1->Q[j], temp\_q2->Q[1]);

}

}

}

temp\_q1 = temp\_q1->next;

if (temp\_q1 == temp\_q3)

break;

}

if (temp\_q1 == temp\_q3)

break;

if (temp\_q1 == NULL)

temp\_b1 = temp\_b1->next;

}

/\*链接\*/

block\* new\_bhead = (block\*)malloc(sizeof(block));

block\* new\_btail = (block\*)malloc(sizeof(block));

new\_bhead->before = new\_bhead;

new\_bhead->q\_head = temp\_q2;

new\_bhead->q\_tail = temp\_b2->q\_tail;

new\_btail->q\_head = temp\_b3->q\_head;

new\_btail->q\_tail = temp\_q3;

temp\_b3->q\_head = temp\_q3->next;

new\_btail->next = head;

head->before = new\_btail;

if (temp\_b3 != temp\_b2->next)

{

new\_bhead->next = temp\_b2->next;

new\_btail->before = temp\_b3->before;

temp\_b3->before->next = new\_btail;

}

else

{

new\_bhead->next = new\_btail;

new\_btail->before = new\_bhead;

}

temp\_b2->next = temp\_b3;

temp\_b3->before = temp\_b2;

temp\_b2->q\_tail = temp\_q2->before;

if (temp\_q2 != temp\_b2->q\_head)

{

temp\_q2->before->next = NULL;

temp\_q2->before = temp\_q2;

}

else

{

temp\_b2->before->next = temp\_b3;

temp\_b3->before = temp\_b2->before;

}

if (temp\_q3->next != NULL)

{

temp\_b3->q\_head = temp\_q3->next;

temp\_q3->next->before = temp\_q3->next;

temp\_q3->next = NULL;

}

else

{

temp\_b3->before->next = NULL;

temp\_b = NULL;

}

head = new\_bhead;

}

/\*代码段生成\*/

void code\_seg(block\* head)

{

FILE\* fp;

fp = fopen("result.txt", "a");

fprintf(fp, "CSEG SEGMENT\nASSUME CS:CSEG,DS:DSEG,SS:SSEG\n\n");

printf("CSEG SEGMENT\nASSUME CS:CSEG,DS:DSEG,SS:SSEG\n\n");

int mark = 0; //段计数

block\* fend\_b = head;

quat\* fend\_q = fend\_b->q\_head;

while (fend\_b->next != NULL)

{

fend\_q = fend\_b->q\_head;

while (fend\_q != NULL)

{

fend\_q = fend\_q->next;

}

if (fend\_q == NULL)

fend\_b = fend\_b->next;

if (fend\_b->next == NULL)

break;

}

fend\_q = fend\_b->q\_tail;

while (fend\_b != head)

{

fend\_q = fend\_b->q\_head;

while (fend\_q != NULL)

{

if (strcmp(fend\_q->Q[0], "f") == 0 && strcmp(fend\_q->Q[1], "end") == 0)

break;

fend\_q = fend\_q->next;

}

if (fend\_q == NULL)

{

fend\_b = fend\_b->before;

continue;

}

if (strcmp(fend\_q->Q[0], "f") == 0 && strcmp(fend\_q->Q[1], "end") == 0)

break;

}

if (fend\_b->next != NULL && fend\_q == fend\_b->q\_tail)

{

fend\_b = fend\_b->next;

fend\_q = fend\_b->q\_head;

}

block\* cur\_block = head;

quat\* cur\_quat = cur\_block->q\_head;

while (cur\_block != NULL)

{

if (cur\_quat == fend\_q && strcmp(fend\_q->Q[1], "end") != 0)

{

fprintf(fp, "\nSTART:MOV AX,DSEG\nMOV DS,AX\nMOV AX,SSEG\nMOV SS,AX\n\n");

printf("\nSTART:MOV AX,DSEG\nMOV DS,AX\nMOV AX,SSEG\nMOV SS,AX\n\n");

}

else if (cur\_quat == fend\_q && strcmp(fend\_q->Q[1], "end") == 0)

{

if (strcmp(fend\_q->before->Q[0], "return") != 0)

{

fprintf(fp, "RET\n");

printf("RET\n");

}

fprintf(fp, "%s ENDP\n", cur\_quat->Q[2]);

printf("%s ENDP\n", cur\_quat->Q[2]);

fprintf(fp, "\nSTART:MOV AX,DSEG\nMOV DS,AX\nMOV AX,SSEG\nMOV SS,AX\n\n");

printf("\nSTART:MOV AX,DSEG\nMOV DS,AX\nMOV AX,SSEG\nMOV SS,AX\n\n");

fprintf(fp, "CALL main\n");

printf("CALL main\n");

break;

}

if (strcmp(cur\_quat->Q[0], "=") == 0)

{

if (cur\_quat->next != NULL && strcmp(cur\_quat->next->Q[0], "for") == 0){}

}

else if (strcmp(cur\_quat->Q[0], ">") == 0 || strcmp(cur\_quat->Q[0], ">=") == 0 || strcmp(cur\_quat->Q[0], "<") == 0 || strcmp(cur\_quat->Q[0], "<=") == 0 || strcmp(cur\_quat->Q[0], "==") == 0)

{

int mark\_tt = mark;

if (strcmp(cur\_quat->next->Q[0], "if") == 0 || strcmp(cur\_quat->next->Q[0], "elif") == 0){}

}

else if (strcmp(cur\_quat->Q[0], "else") == 0)

{

cur\_quat = cur\_quat->next;

continue;

}

else if (strcmp(cur\_quat->Q[0], "ie") == 0){}

else if (strcmp(cur\_quat->Q[0], "for") == 0 || strcmp(cur\_quat->Q[0], "while") == 0)

{

if (strcmp(cur\_quat->Q[0], "for") == 0)

{

block\* b = cur\_block->next;

quat\* q = b->q\_head;

if (q->Q[2][0] <= '0' || q->Q[2][0] >= '9')

{

fprintf(fp, "MOV BX,%s\nDEC BX\nMOV %s,BX\n", q->Q[2], q->Q[2]);

printf("MOV BX,%s\nDEC BX\nMOV %s,BX\n", q->Q[2], q->Q[2]);

}

else

{

int temp = atoi(q->Q[2]);

memset(q->Q[2], '0', sizeof(q->Q[2]));

itoa(temp, q->Q[2], 10);

}

}

mark++;

if (strcmp(cur\_quat->Q[0], "while") == 0)

{

int i = 0;

for (i = 0; i < 20; i++)

{

if (count\_while[i] == 0)

break;

}

count\_while[i] = mark;

}

else if (strcmp(cur\_quat->Q[0], "for") == 0)

{

int i = 0;

for (i = 0; i < 20; i++)

{

if (count\_for[i] == 0)

break;

}

count\_for[i] = mark;

}

}

else if (strcmp(cur\_quat->Q[0], "fe") == 0 || strcmp(cur\_quat->Q[0], "we") == 0)

{

if (mark > 0)

{

if (strcmp(cur\_quat->Q[0], "fe") == 0)

{

int i = 19;

for (i; i >= 0; i--)

{

if (count\_for[i] != 0)

break;

}

fprintf(fp, "JMP block%d\n", count\_for[i]);

printf("JMP block%d\n", count\_for[i]);

count\_for[i] = 0;

}

else if (strcmp(cur\_quat->Q[0], "we") == 0)

{

int i = 19;

for (i; i >= 0; i--)

{

if (count\_while[i] != 0)

break;

}

fprintf(fp, "JMP block%d\n", count\_while[i]);

printf("JMP block%d\n", count\_while[i]);

count\_while[i] = 0;

}

}

}

else if (strcmp(cur\_quat->Q[0], "+") == 0){}

else if (strcmp(cur\_quat->Q[0], "-") == 0){}

else if (strcmp(cur\_quat->Q[0], "\*") == 0){}

else if (strcmp(cur\_quat->Q[0], "/") == 0){}

else if (strcmp(cur\_quat->Q[0], "++") == 0){}

else if (strcmp(cur\_quat->Q[0], "--") == 0){}

else if (strcmp(cur\_quat->Q[0], "f") == 0) //函数

{

if (strcmp(cur\_quat->Q[1], "end") != 0)

{

fprintf(fp, "%s PROC NEAR\n", cur\_quat->Q[1]);

printf("%s PROC NEAR\n", cur\_quat->Q[1]);

}

else

{

if (strcmp(cur\_quat->before->Q[0], "return") != 0)

{

fprintf(fp, "RET\n");

printf("RET\n");

}

fprintf(fp, "%s ENDP\n\n", cur\_quat->Q[2]);

printf("%s ENDP\n\n", cur\_quat->Q[2]);

}

if (strcmp(cur\_quat->Q[1], "end") != 0)

{

block\* temp\_endb = cur\_block;

quat\* temp\_endq = cur\_quat;

while (temp\_endb != NULL)

{

temp\_endq = temp\_endb->q\_head;

while (temp\_endq != NULL)

{

if (strcmp(temp\_endq->Q[1], "end") == 0)

{

break;

}

temp\_endq = temp\_endq->next;

}

if (temp\_endq == NULL)

{

temp\_endb = temp\_endb->next;

continue;

}

if (strcmp(temp\_endq->Q[1], "end") == 0)

break;

}

block\* tempb\_digui = cur\_block;

quat\* tempq\_digui = cur\_quat;

while (tempb\_digui != temp\_endb->next)

{

tempq\_digui = tempb\_digui->q\_head;

while (tempq\_digui != NULL)

{

if (strcmp(tempq\_digui->Q[0], "call") == 0 && strcmp(tempq\_digui->Q[1], cur\_quat->Q[1]) == 0)

{

break;

}

tempq\_digui = tempq\_digui->next;

}

if (tempq\_digui == NULL)

{

tempb\_digui = tempb\_digui->next;

continue;

}

if (strcmp(tempq\_digui->Q[0], "call") == 0 && strcmp(tempq\_digui->Q[1], cur\_quat->Q[1]) == 0)

{

break;

}

}

if (tempq\_digui != NULL && strcmp(tempq\_digui->Q[0], "call") == 0 && strcmp(tempq\_digui->Q[1], cur\_quat->Q[1]) == 0)

{

strcpy(digui[digui\_num], cur\_quat->Q[1]);

digui\_num++;

block\* temp\_b = head;

quat\* temp\_q = temp\_b->q\_head;

while (temp\_b != NULL)

{

temp\_q = temp\_b->q\_head;

while (temp\_q != NULL)

{

if (strcmp(temp\_q->Q[0], "declarP") == 0 && strcmp(temp\_q->Q[1], cur\_quat->Q[1]) == 0)

{

break;

}

temp\_q = temp\_q->next;

}

if (temp\_q == NULL)

{

temp\_b = temp\_b->next;

continue;

}

if (strcmp(temp\_q->Q[0], "declarP") == 0 && strcmp(temp\_q->Q[1], cur\_quat->Q[1]) == 0)

{

break;

}

}

char a[5][20] = { 0 };

int k = 0;

int i = 1;

char c;

c = temp\_q->Q[3][1];

while (c != ')')

{

int j = 0;

while (c != '|' && c != ')')

{

a[k][j] = c;

i++;

j++;

c = temp\_q->Q[3][i];

if (c == '|' || c == ')')

{

break;

}

}

k++;

if (c == ')')

{

break;

}

i++;

c = temp\_q->Q[3][i];

}

for (int n = 0; n < k; n++)

{

fprintf(fp, "PUSH %s\n", a[n]);

printf("PUSH %s\n", a[n]);

}

}

}

}

else if (strcmp(cur\_quat->Q[0], "return") == 0){}

else if (strcmp(cur\_quat->Q[0], "call") == 0){}

if (cur\_quat->next != NULL)

{

cur\_quat = cur\_quat->next;

}

else

{

mark++;

fprintf(fp, "block%d:", mark);

printf("block%d:", mark);

if (cur\_block->next != NULL)

{

cur\_block = cur\_block->next;

cur\_quat = cur\_block->q\_head;

}

else

{

fprintf(fp, "CALL main\n");

printf("CALL main\n");

break;

}

}

}

fprintf(fp, "\nMOV AH,4CH\nINT 21H\nCSEG ENDS\nEND START\n");

printf("\nMOV AH,4CH\nINT 21H\nCSEG ENDS\nEND START\n");

fclose(fp);

}

**5.2.4 目标代码生成样例测试结果**

此次目标代码生成，最终结果为输出可直接运行的8086汇编代码

可实现的功能有：

变量声明及定义；

算术运算表达式；

赋值语句；

数组的声明、赋值和调用；

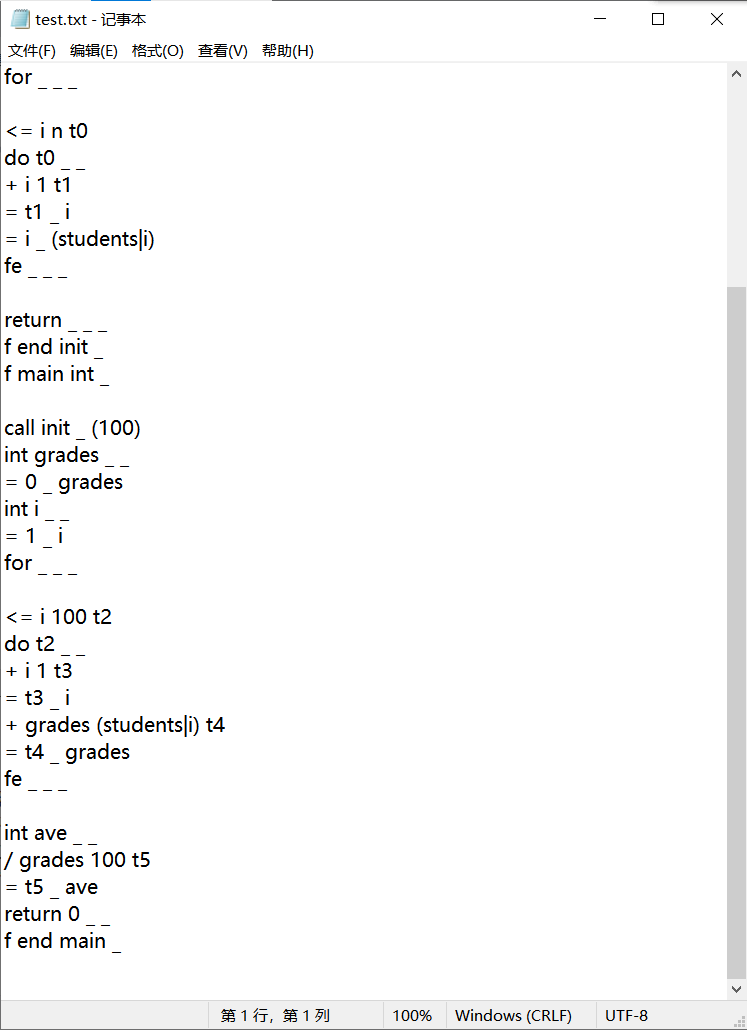
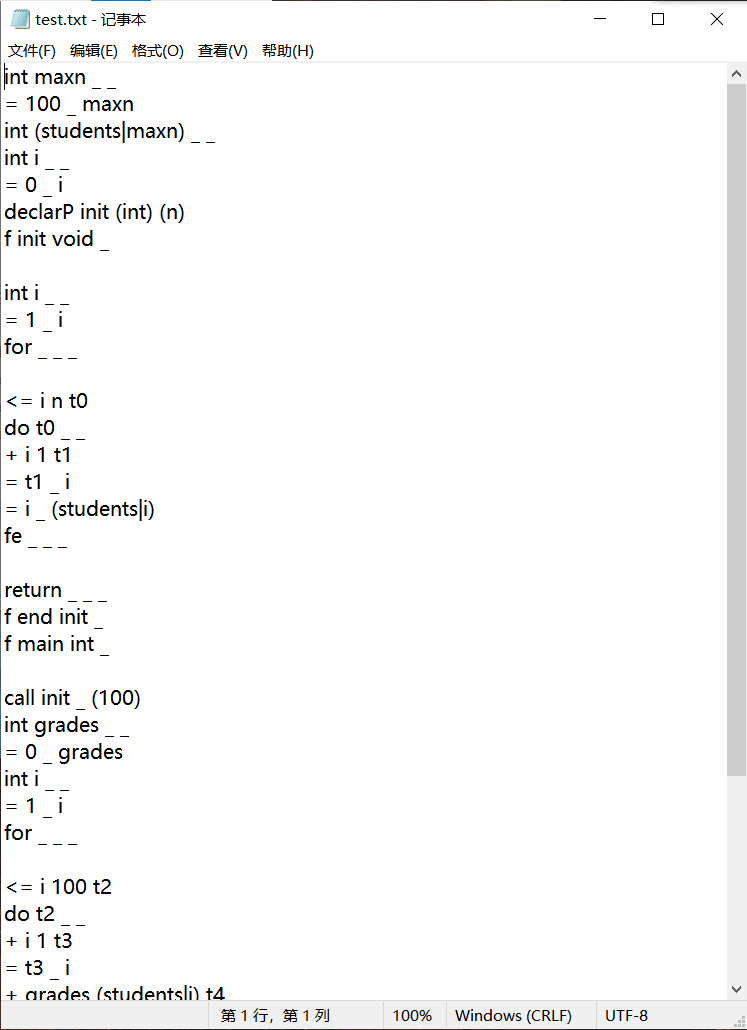
If 语句；

While 语句；

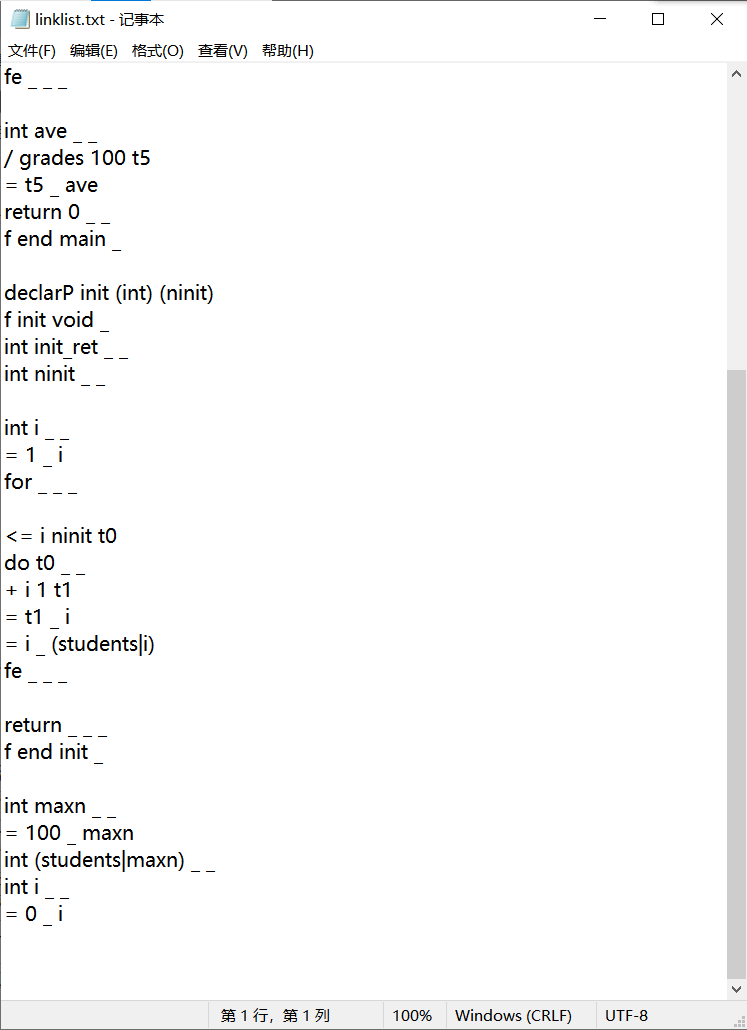
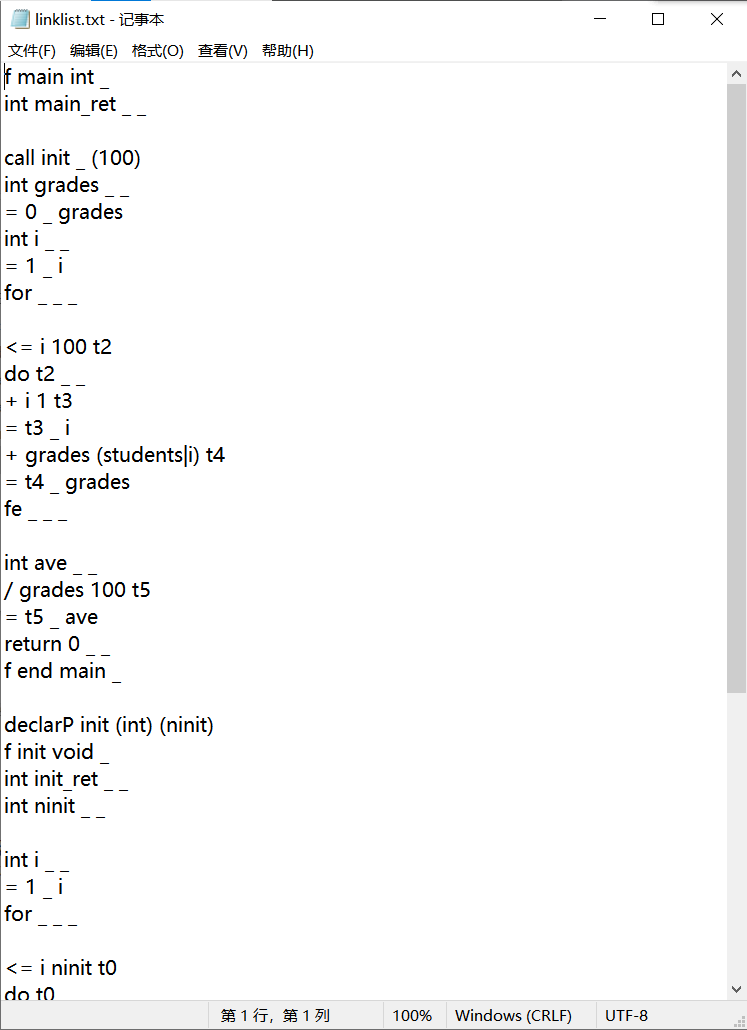
For语句；  
 允许嵌套与递归的可含参函数的声明与调用；

If，While，For语句，函数之间的任意多重嵌套

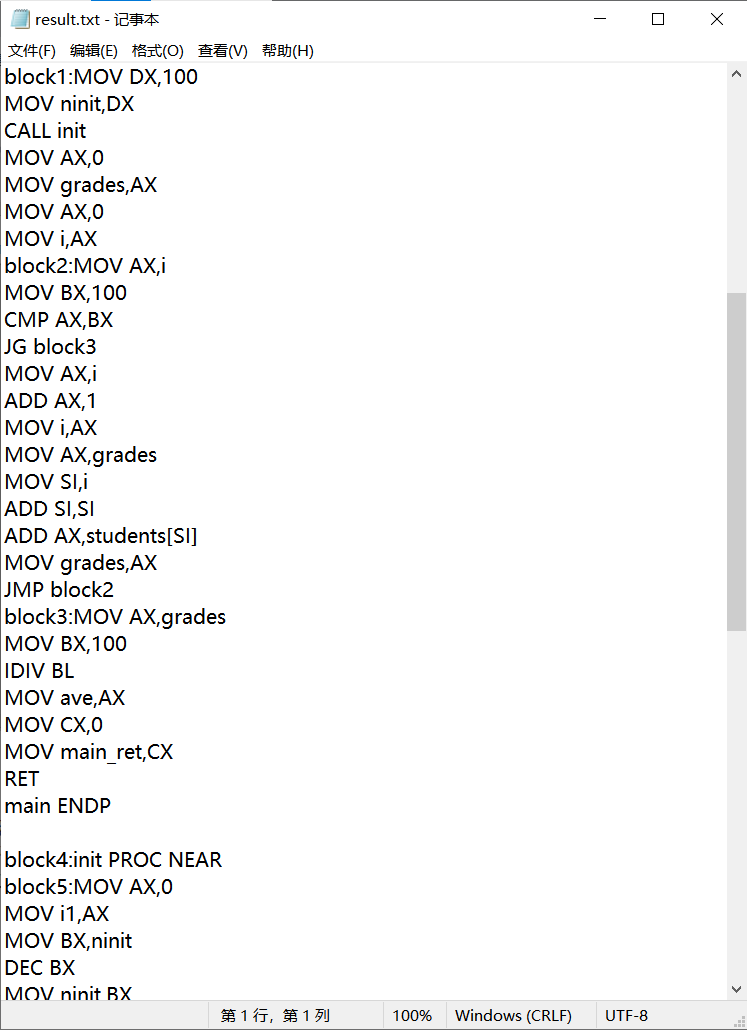
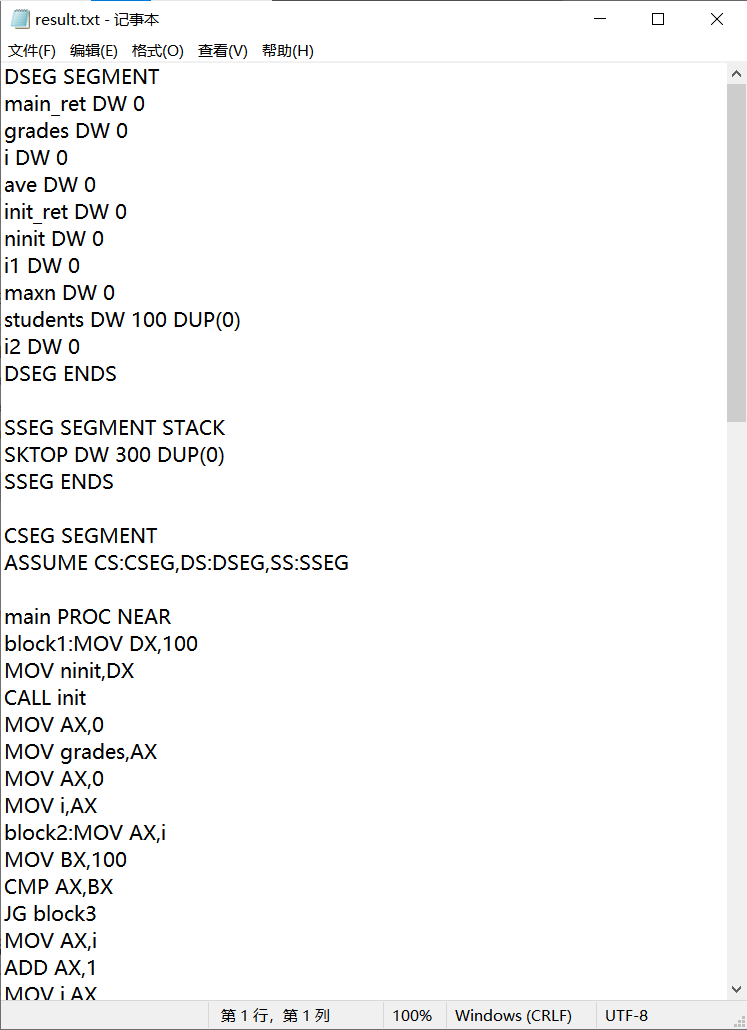
测试文件（四元式形式）：

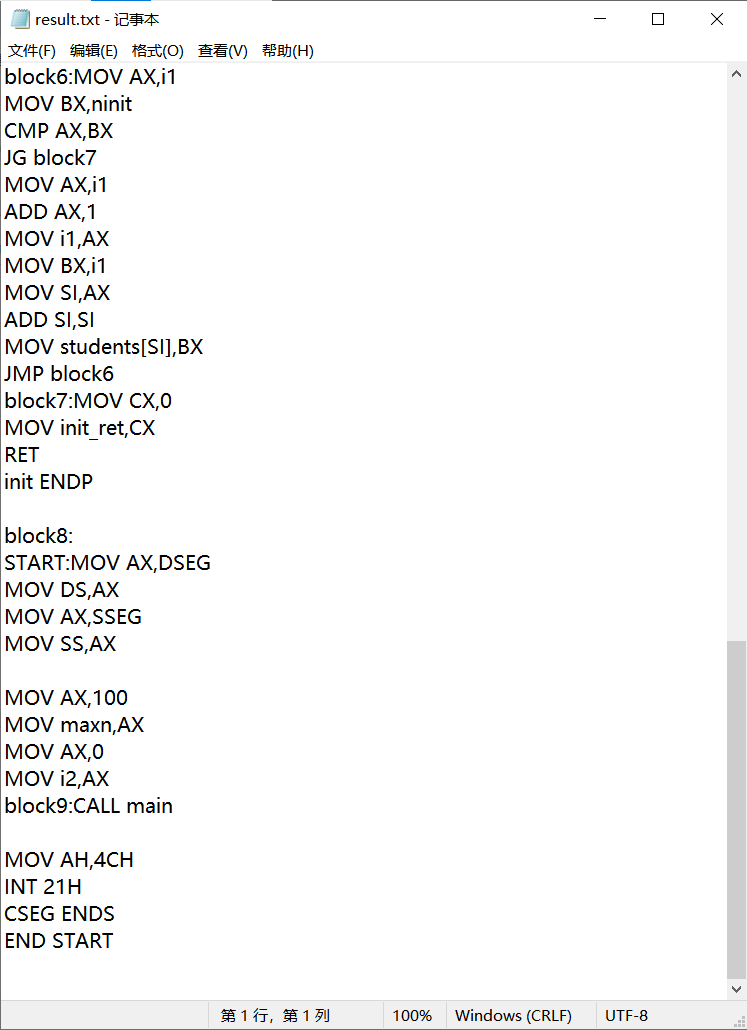


初步处理后链表内容：

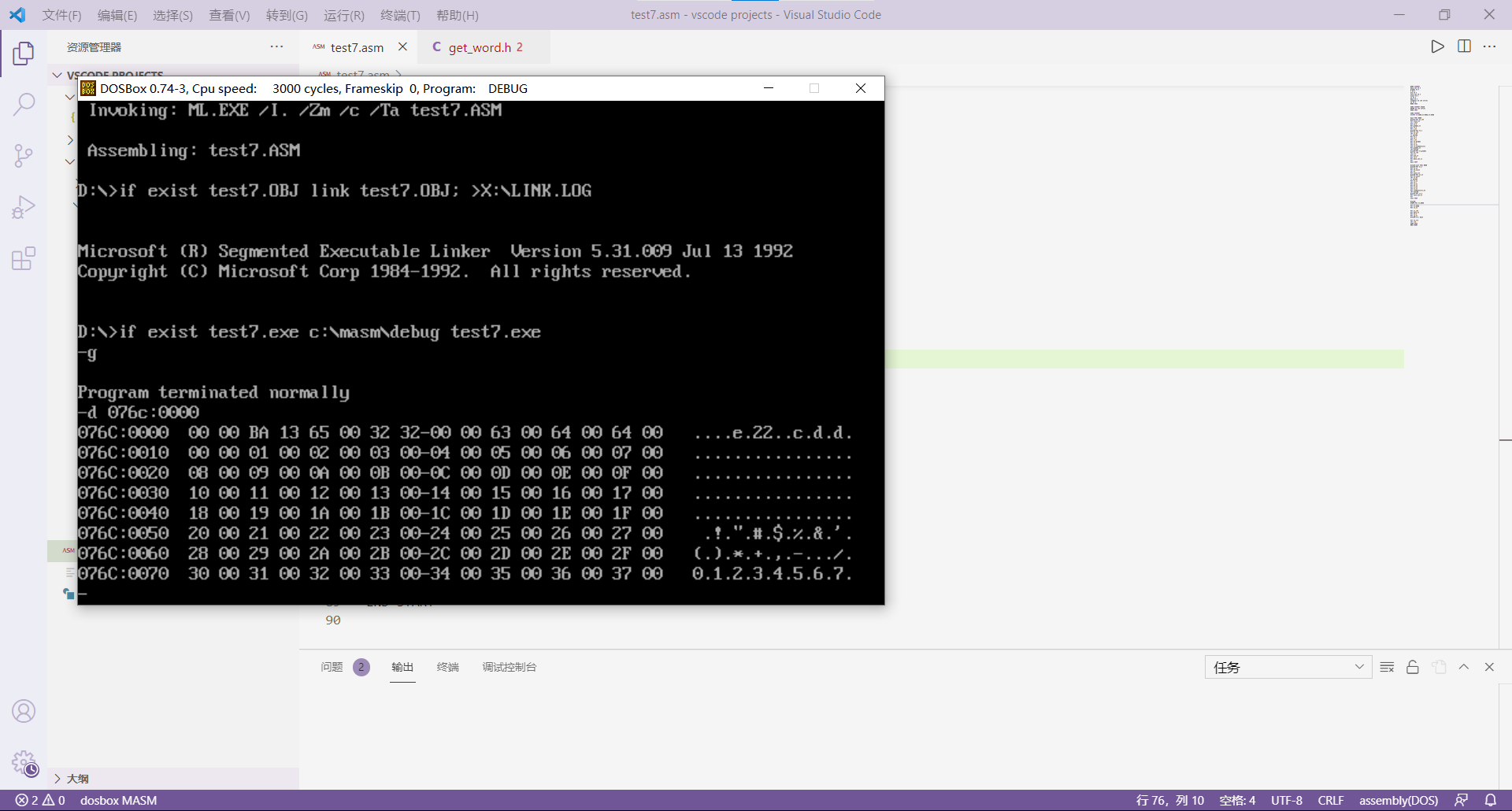


运行结果（输出可直接运行的汇编代码）：



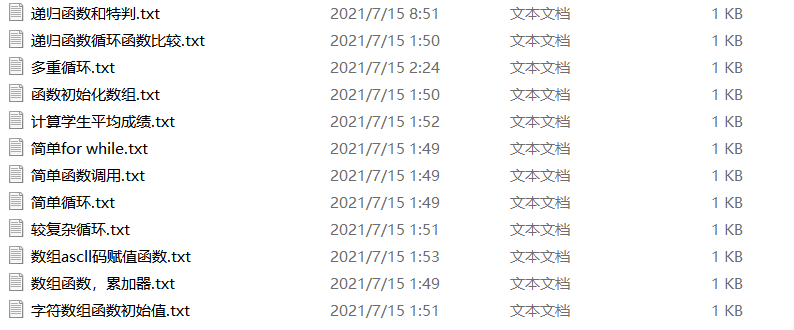


汇编代码运行结果：



可以看到，grades变量在程序运行结束后数值为13ba，即5050，与原C程序输出数值一致，该例目标代码生成成功

下图为一些通过的测试样例，目标代码运行结果均与源程序输出一致



**6 结论**

编译器包含词法分析、语法分析、语义分析（符号表系统和中间代码生成）、 中间代码的简单优化和目标代码的生成。此次我们做的编译器基本实现了C语言文法(部分为python文法)，文法包括变量和常量的声明，包括int，bool，char等声明，以及数组、待参函数的声明和调用，还包括函数的递归调用，算术运算，逻辑运算，赋值语句，条件选择语句(if、elif、else)， 循环（for,while）语句。利用自动机和递归下降子程序的方法，插入相应语义操作，结合相应文法可以实现一个编译器的设计。词法分析部分包含识别单词和生成 Token 两部分.读入采用读文件的方式读入源文件，源程序经过词法分析器后生成了 Token 序列输出到文件中，便于以后的四元式生成。语法分析部分使用递归下降子程序文法，语义分析即是在递归过程中插入语义动作，包括查填符号表以及生成四元式，并借助符号表系统进行各类语义检查，做报错处理等，将四元式中间代码交于编译器后端。四元式在后端经过简单优化，对代码进行等价变换，使变换后的代码具有更高的时间和空间效率后交付于目标代码生成器。

目标代码生成器将简单优化过的四元式中间代码序列生成多寄存器下经过寄存器分配优化的目标代码序列。通过以上过程即可实现一个简单的比较完整的编译器。

**7 参考文献**

1、陈火旺.*《程序设计语言编译原理》（第3版）*. 北京：国防工业出版社.2000.

2、美 Alfred V.Aho Ravi Sethi Jeffrey D. Ullman著.李建中，姜守旭译.《*编译原理*》.北京：机械工业出版社.2003.

3、美 Kenneth C.Louden著.冯博琴等译.《*编译原理及实践*》.北京：机械工业出版社.2002.

4、金成植著.《*编译程序构造原理和实现技术*》. 北京：高等教育出版社. 2002.

**8 收获，体会和建议**

**伍孝飞**：通过进行本次前端的的设计，使我能够更好地掌握编译原理的基本理论和编译程序构造的基本方法和技巧，融会贯通本课程所学专业理论知识，培养我独立分析问题、解决问题的能力，以及系统软件设计的能力，培养我的创新能力及团队协作精神。同时极大地加深了我对前端的理解，以及增强了我的代码书写能力，对C语言的语法更加清晰。

同时感谢老师以及学长在我们编写代码过程中的指导，老师的指导使我受益匪浅，老师的指导解决了我的疑问，提高了我代码的书写效率。

**熊程林：**这次我的任务是四元式生成以及四元式的简单优化，这两部分在课上进行学习的时候我就发现了它们对于编译器的重要性，这次也是我第一次尝试自己编写编译器，在对自己负责的模块编写过程中，我也遇到了不少问题，同时也学到了很多新的知识，加强我对代码的理解，对递归，以及栈的使用更加的熟悉和了解，以及自动机的理解和使用。

经过这次课程设计，我对编译原理课程的理解更加深刻，同时自己的编程能力也得到了锻炼，提升了自己思考问题，分析问题，处理问题，解决问题的能力，让我受益匪浅。

**唐知宸：**这次实验中我主要负责目标代码生成，在实验进行过程中，我也发现了自己存在的不足和问题。我们的基本思路是从简单文法开始到逐步增加复杂文法。开始的简单文法部分的目标代码生成我还得心应手，但是到了后期，增加了函数的调用，数组的使用，也让目标代码生成的难度极大的提高，代码量也从开始的一千行以内增加到了三千行，这个过程中不仅加深了我对编译器的理解，也同时让我对汇编语言更加了解，熟悉。

这次课程设计的挑战还是不小的，同时在团队配合中我也明白了合理分配的重要性，对我今后的学习有很大帮助。