# 2020编译原理设计文档

18373441

覃启浩

# 词法分析阶段

## 题目要求

请根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值。

| 单词名称 | 类别码 | 单词名称 | 类别码 | 单词名称 | 类别码 | 单词名称 | 类别码 |
|---|---|---|---|---|---|---|---|
| 标识符 | IDENFR | else | ELSETK | - | MINU | = | ASSIGN |
| 整形常量 | INTCON | switch | SWITCHTK | * | MULT | ; | SEMICN |
| 字符常量 | CHARCON | case | CASETK | / | DIV | , | COMMA |
| 字符串 | STRCON | default | DEFAULTTK | < | LSS | ( | LPARENT |
| const | CONSTTK | while | WHILETK | <= | LEQ | ) | RPARENT |
| int | INTTK | for | FORTK | > | GRE | [ | LBRACK |
| char | CHARTK | scanf | SCANFTK | >= | GEQ | ] | RBRACK |
| void | VOIDTK | printf | PRINTFTK | == | EQL | { | LBRACE |
| main | MAINTK | return | RETURNTK | != | NEQ | } | RBRACE |
| if | IFTK | + | PLUS | : | COLON | | |

## 实现过程

### 将单词分类

1. 单字符

   +-*/:;,()[]{}

   直接判断即可

   还有一个!=也是可以直接判断的，但不属于单字符

2. 单字符后面可能会有不同情况

   < <=  > >=  = ==

3. 字母或数字串

1. const int...return
2. 标识符

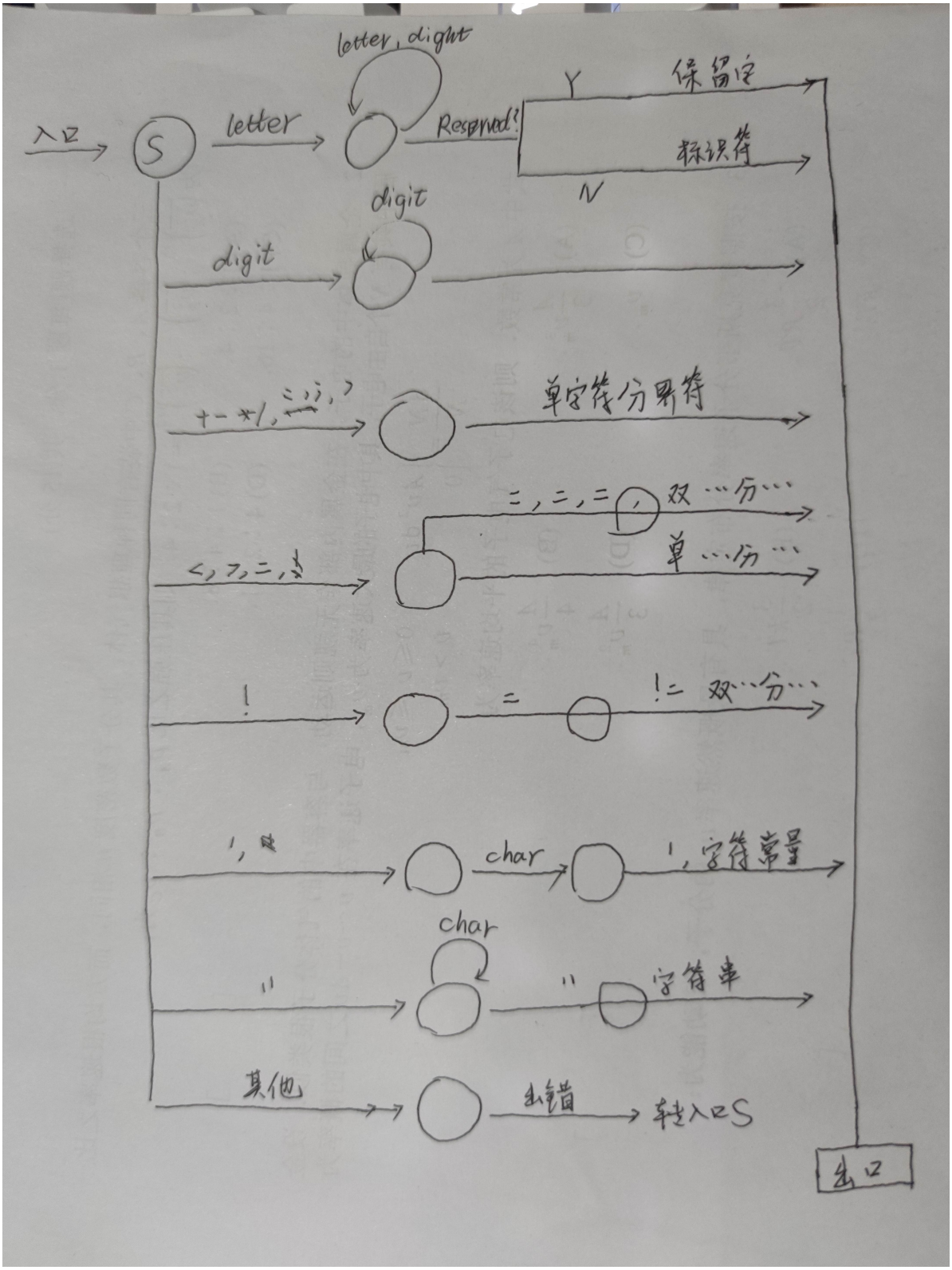
需要判断是否为标识符，用一个reserver函数就能解决

4. 整形变量

简单

5. 字符常量和字符串

## 画出状态图




## 编写程序并调试

### 编程实现

#### 主要函数

```
void getChar()
void clearToken()

//判断字符类型
bool isNewLine()
bool isWhitespace()
```

```
bool isBar()
bool isLetter()
bool isDigit()
bool isPlus()
bool isMinus()
bool isMult()
bool isDivi()
bool isLess()
bool isGreater()
bool isEqual()
bool isNot()
bool isColon()
bool isSemi()
bool isComma()
bool isLpar()
bool isRpar()
bool isLbrack()
bool isRbrack()
bool isLbrace()
bool isRbrace()
bool isOneQuote()
bool isTwoQuote()

void catToken()
void retract()
void reserver()
//将数字类型转换为int
int transNum(string token)
//读取下一个symbol
int getsym()
//输出symbol
string symbolStr()
```

**数据结构**

```
class Token
{
public:
    static int pos;
    static int count;
    string str;
    Symbol symbol;
    int line;

public:
    Token();
    Token(string str, Symbol symbol, int line);
};

vector<Token> words;
```

### 遇到的bug

第一个bug是一个很智障的bug，我自己写了getChar函数来从文件中读取一个字符，就是把文件中读到的字符赋值给全局变量theChar来记录，所以getChar应该是void类型的，我却写成了char类型导致没有返回值，但是程序竟然是能运行的并且没有报错，所以提交了很多次都是runtime error。

第二个bug应该是写代码时复制粘贴错了，whiteSpace忘了回车符和制表符，大于号弄成了小于号。

## 总结

这次实验难度并不是很大，并且由于史晓华老师在之前就要求我们实现了一个教材上的语法分析器，我在拿到题目的时候就知道怎样去做了。这也是我真正第一次用上c++，我希望以后的难度也不会特别大，至少只要我认真地学了，认真地做了，就不会挂科的。

# 语法分析阶段

## 题目要求

根据给定的文法设计并实现语法分析程序，能基于上次作业的词法分析程序所识别出的单词，识别出各类语法成分。

## 设计思路

在编译原理课上，我学到了**递归下降**的分析方法，对每个非终结成分都编写出一个子程序，来该成分所对应的语法分析的分析和识别成分。编成递归子程序的原因是，文法是具有递归性的。

具体到**代码实现**上时，就是每次读取一个单词，然后根据预读到的单词进行下一步或者进入子程序，用if语句while语句来控制。我使用了一个数组来存储所有的单词来预读以解决回溯问题，甚至如果预读1个不能解决那就预读更多个来解决，根据预读到的单词判断应该进入哪个子程序中分析。

## 编程实现

### 数据结构

```
//单词类别
enum symbolTypes
{
    CONST, VAR, ARRAY, FUNC
};
//数据类型
enum dataTypes
{
    INT, CHAR, VOID
};
//符号表中的一个元素
struct Entry
{
    int symType;    /*const var func array*/
    int dataType;   /*
                    for const and var:int or char
                    for ret_func: int or char
                    for no_ret_func: void
                    */
    int value;      /*
                    for int and char: 不用管了
```

```
                    for array:its max length
                    for func: whether it is a leaf function? 什么是叶子函数，这里是优化的时候用到的吗？
                    */

    int offset;     //address, aka offset of base address
                    /*
                    * 对于全局变量来说，应该是相对于$gp的偏移
                    * 对于局部变量来说，应该是相对于$fp的偏移
                    * 对于函数来说，应该是所有本地的变量和和数组加起来的大小，这里的本地变量包括temp吗？
                    */
};
//函数参数
struct Para
{
    int dataType;
    string name;
};
```

## 主要函数

```
//语法分析的一堆函数
string stringgg();

void program();

void constDescrip();

void constDefine();

int unsignedInt();

int integer();

void declareHead();

int constant(int * dataType=NULL);

void varDescrip();

void varDefine();

void noInitVarDefine();


void initVarDefine();


void retFuncDefine();

void noRetFuncDefine();

void compStatement();

void paraTable(string funcName);
```

```
void mainFunc();

int expression(string* opd = NULL);

int item(string* opd = NULL);

int factor(string* opd = NULL);

void statement();

void assignStatement();

void condStatement();

void condition(string label="", bool oppo=true);

void loopStatement();

int pace();

void caseStatement();

void caseTable(int exprType);

void caseSubStatement(int exprType);

void defaulttt();

void retFuncCall(string* opd = NULL);

void noRetFuncCall();

void valueParaTable(string funcName);

void statementList();

void readStatement();

void writeStatement();

int returnStatement();
```

## 重点难点

这次实验思路上并不是特别难，通篇的难点在于**繁琐的细节**和过多的语法成分子程序。这次实验我花了18个小时左右，其中有一半的时间是花在子程序的debug上的。

第一个需要解决的问题是如何**预读**。最初我是一边从文件中读取单词，一边分析单词同时根据已经读到的单词分析语法，这样做的问题是如果不确定子程序是什么，就需要预读，而预读就需要进行词法分析，词法分析就会输出读到的单词，这还没开始分析单词就输出了肯定不行。于是我建了一个类来存储分析到的单词，同时建了一个数组存储类，语法分析时想怎么预读就怎么预读，非常方便。

第二个问题是某一些可以为**空的子程序**带来的问题。可以为空的子程序与其他程序不同，如果为空，那么就需要判断为空的条件也就是下一个单词是什么（大多为')'和'}'），而且这个子程序不能像其他程序一样读取下一个单词，因为下一个单词不是它的，而是它的上层程序的，上层程序需要这个下一个单词来判断程序是否正确。这个问题是我到最后才弄清楚的。

## 不足之处

**冗长的代码。**这次代码语法分析这块我写了将近1300行代码，其中有很多是相似的代码段，考虑到后面可能会有不同的需求，我没有用宏来整理这些相似甚至相同的代码段。还有就是我用到了很多的if else嵌套结构，显得有点混乱，影响我的思路，或许我还能再优化一下。

**优化不足。**我现在并没有考虑任何有关优化的事，做的只是为了能把能通过测试，这样有好处也有坏处，好处是不用想太多，坏处是后期优化起来可能会比较不方便。

**对c++不熟悉。**说实话这是我第一次真正拿c++编程，我对很多用法对不太熟悉，可能错过了更好的写法和更优的解法，但是目前也就这样吧，我只要在学习中进步就行了。

## 总结

总的来说，这次实验花了我挺多精力的，不是说它难以理解，而是繁复冗杂，最后花了我很多时间。这就才只是第三次实验，我觉得我还有很长的路要走，后面花的时间可能会更多，且行且看吧，加油。

# 错误处理阶段

## 题目要求

据给定的文法设计并实现错误处理程序，能诊察出常见的语法和语义错误，进行错误局部化处理，并输出错误信息。

## 设计思路

#### 建立符号表

在编译原理课上，我学习到了错误处理和符号表的相关知识，在经过一段时间思考之后，我对于这次实验的完成有了大致思路。

首先是**建立符号表**，这是最基本和最重要的。有了符号表，就可以存取各种变量和函数的信息，就可以判断根据符号表中的信息判断是否有错误了。我把符号表分为全局符号表（包括全局变量和函数）以及局部符号表（包括局部变量），还有一个函数的参数表，开始我以为会出现多层结构的，后面经过分析之后发现局部变量只会有一层，那就只用建立一个局部符号表就够了。

初次分析的时候，我认为符号表中存的数据应当包含如下信息：

- 变量名及其类型
- 常量名及其类型
- 函数的参数个数以及类型
- 函数名及是否有返回值
- 数组的名字及其维数

可以把符号表中的数据分为四种类型：变量，常量，函数，数组

#### 分类解决错误

在大致理清思路之后，我开始着手解决题目要求了。

我首先将所以的错误分类，从简单到复杂，一步步解决问题：

| 难度和复杂程度 | 错误类型 |
| --- | --- |
| 无脑简单 | 缺少分号<br>缺少括号<br>缺少缺省 |
| 比较简单，基本不需要符号表 | 非法符号<br>函数是否有返回<br>数组初始化个数<br>条件判断 |
| 需要涉及符号表 | 剩余 |

# 编程实现

## 数据结构

```
//错误类型的枚举
enum errorType
{
    ILLEGAL_CHAR,//a
    REDEFINE_NAME, //b
    UNDEFINE_NAME,//c
    FUNC_PARA_NUM_NOT_MATCH,//d
    FUNC_PARA_TYPE_NOT_MATCH,//e
    ILLEGAL_CONDITION,//f
    ERROR_RET_IN_VOID_FUNC,//g
    ERROR_RET_IN_RET_FUNC,//h
    INDEX_NOT_INT,//i
    TO_CHANGE_CONST,//j
    SHOULD_BE_SEMI,//k
    SHOULD_BE_RPARENT,//l
    SHOULD_BE_RBRACK,//m
    ARRAY_COUNT_ERROR,//n
    CONST_TYPE_ERROR,//o
    MISS_DEFAULT//p
};
```

## 符号表

数据结构

```
//单词类别
enum symbolTypes
{
    CONST, VAR, ARRAY, FUNC
};
//数据类型
enum dataTypes
{
    INT, CHAR, VOID
};
//符号表中的一个元素
struct Entry
{
```

```
    int symType;    /*const var func array*/
    int dataType;   /*
                    for const and var:int or char
                    for ret_func: int or char
                    for no_ret_func: void
                    */
    int value;      /*
                    for int and char: 不用管
                    for array:its max length
                    */

    int offset;     //address, aka offset of base address
                    /*
                    * 对于全局变量来说，应该是相对于$gp的偏移
                    * 对于局部变量来说，应该是相对于$sp的偏移
                    * 对于函数来说，应该是所有本地的变量和和数组加起来的大小
                    */
};
//函数参数
struct Para
{
    int dataType;
    string name;
};
```

符号表(通过hashmap实现)

```
//符号表，函数参数表
map<string, Entry> globalSymTable;
map<string, Entry> localSymTable;
map<string, vector<Para>> funcParaTable;
//存放所有的本地符号表，以函数名字为key键值
map<string, map<string, Entry>> allLocalSymTable;
```

### 主要函数

```
bool inGlobalSymTable(string name)
bool inLocalSymTable(string name)
bool inTable(string name)
bool inAllLocalTable(string name)
void insertGlobalTable(string name, Entry entry)
void insertLocalTable(string name, Entry entry)
void insertTable(string name, Entry entry)
void clearLocalTable(string name = "")
```

## 重点难点

**架构的确立。**这个架构指的是用什么来完成实验，是用类还是用结构体，是用全局变量还是用函数传递参数，错误处理的那些函数和定义声明应该放在哪些地方，这些问题都确实困扰到了我。

符号表的建立。应该有哪些符号表，符号表中存哪一些数据，仔细分析之后我用结构体存储符号表中的一个元素，同时用hashmap建立名字到符号表结构体的索引。

如何**具体解决每一种错误类型。**每一种错误类型的解决办法是不一样的，比如空的字符或者非法字符这类错误就需要在词法分析读取字符或字符串时就发现，而缺少分号这种类型就是在语法分析时简单地判断symbol是否是分号就行了，而函数的参数类型和个数是否满足则需要查找建立的符号表来判断是否有错误，这些不同类型的错误分布在语法分析的各种地方，这些情况不是统一的，它们有时候让我头绪混乱。

**修改原有代码。**在有错误处理的情况下，由于错误的存在，我们需要对语法分析甚至是词法分析代码进行修改，比如预读可能需要调整，很多分析子程序的函数类型可能需要从void变为int等类型，不同函数之间可能需要借用全局变量或者传递参数，比如表达式、项、因子需要有返回类型来判断是int类型还是char类型。这还是花费了我不少的精力。

**debug。**我个人是没有太多的bug的，在群友的帮助下，我比较轻松的解决了一个80%的问题，而有些人就头疼了好久，如果没有大家的帮助，如何构建测试数据以及如何找出存在的bug绝对是个不小的挑战。

## 不足之处

**架构不好。**我总觉得我的代码架构设计有很多不好的地方，或者说写得不够规范，比如说全局变量的泛滥，比如大括号有时候有有时候无，比如说存在很多冗余无用效率低的代码，这些我觉得可能都是可以改进的，但是我不知道怎样才是最规范的，目前来说，完成实验已经很好了。

## 总结

这次实验花的时间跟语法分析基本一样，这其实说明了我效率有点低下，喜欢在学习的时候分心。总的来说，我比较成功地完成了这次实验，后面的日子，加油吧。

# 代码生成阶段

## 题目要求

请在词法分析、语法分析及错误处理作业的基础上，为编译器实现语义分析、代码生成功能。

完成编译器，将源文件（统一命名为testfile.txt）编译生成MIPS汇编并输出到文件（统一命名为mips.txt）。

按如上要求将目标代码生成结果输出至mips.txt中，中文字符的编码格式要求是UTF-8。

## 思路设计

### 代码生成第一次作业

第一次作业包括比较少的语法成分。

第一次作业包括以下部分：

- 变量和变量说明
  - 常量说明
  - 变量说明
- 简单的语句
  - 读语句
  - 写语句
  - 赋值语句
- 与计算有关
  - 表达式
  - 项

    - 因子

首先要做的是生成中间代码，也就是四元式。四元式中间几乎包括了源代码中的所有信息。

有了四元式，我们就可以生成目标代码了，我们还需要解决程序运行时的一些问题：全局变量如何存取，局部变量如何存取，临时变量如何存取，如何获得他们的地址。

我的解决策略是这样的：

全局变量存在.data段，可以首先让gp指针位于.data段的初始地址，这样我们就可以直接通过gp加上全局变量的偏移存取了。

局部变量存在sp中，sp以下的地址存局部变量，通过sp加上局部变量的偏移存取。

临时变量当做局部变量，和局部变量一起存放在sp以下的内存中。

## 代码生成第二次作业

第二次作业需要在第一次作业的基础上完善，完成所有的语法成分。

第二次作业包括以下部分：

- 数组有关
    - 常量定义
    - 变量定义
    - 因子
- 函数有关
    - 函数定义
    - 函数调用
- 其他语句
    - 条件语句
    - 循环语句
    - 情况语句

为了解决数组的问题，我引入了两个中间代码指令:

- ARRAYPUT
    - a[]=t，给数组元素赋值
- ARRAYGET
    - t=a[]，从数组中取值

所有的数组都是连续地址空间，所以中间代码和目标代码中，一维数组和二维数组都被当成一维数组。

二维数组需要先通过两个下标计算出相对位置，再通过相对偏移访问内存来存取。

几种语句的实现大同小异，都是通过加入跳转语句和分支语句来实现的。

函数的实现相对困难，函数在调用前需要保存环境（包括ra寄存器，sp寄存器，参数，局部变量，以及需要保存的寄存器）。

在我的实现中，sp用来存取局部变量，fp用来存放函数参数。

## 编程实现

中间代码指令：

```
enum operation
{
    LABEL,//:

    PLUS_OP,MINU_OP,MULT_OP,DIV_OP,//+,-,*,/
    ASSIGN_OP,//=

    //<,>...
    LSS_OP,
    LEQ_OP,
    GRE_OP,
    GEQ_OP,
    EQL_OP,
    NEQ_OP,

    JUMP,

    PUSH,//函数调用时参数传递
    CALL,//函数调用
    RET,//函数返回

    //用readi来判断是读int型还是char型，print同理
    READI,
    READC,
    PRINTI,
    PRINTC,
    PRINTS,

    //关于数组
    GETARRAY,//取数组的值,t=a[]
    PUTARRAY,//给数组元素赋值,a[]=t

    EXIT  //退出程序
};
```

中间代码class:

```
class interCode
{
public:
    operation op;
    string z;//结果
    string x, y;//操作数

    interCode(int opp, string zz = "", string xx = "", string yy = "");
};
```

mips代码指令：

```
enum mipsOpretion
{
    add,addi,addu,addiu,
```

```
    sub,subi,subu,subiu,
    lui,
    multop,mul,
    divop,
    mfhi,mflo,
    moveop,

    sll,srl,

    li,la,

    beq,
    bne,
    bgt,
    bge,
    blt,
    ble,
    blez,
    bgtz,
    bgez,
    bltz,

    j,
    jal,
    jr,

    lw,
    sw,

    syscall,

    dataSeg,
    textSeg,
    spaceSeg,
    asciizSeg,
    globalSeg,
    label,
};
```

mips代码类:

```
//所有操作数按照mips顺序存放
class mipsCode
{
public:
    mipsOpretion op;
    string z;
    string x;
    string y;
    mipsCode(mipsOpretion opp, string zz = "", string xx = "", string yy = "");
};
```

mips所有寄存器

```
const string regs[] = {
    "$zero",
    "$v0", "$v1",
    "$a0", "$a1", "$a2", "$a3",
    "$t0", "$t1", "$t2", "$t3", "$t4", "$t5", "$t6", "$t7", "$t8", "$t9",
    "$s0", "$s1", "$s2", "$s3", "$s4", "$s5", "$s6", "$s7",
    "$gp", "$sp", "$fp",
    "$ra"
};
```

生成目标代码主要函数：

```
//把name的值写入寄存器regStr
void loadValue(string name, string& regStr);
//把regStr中的值写入name所在的地址，这个地址可能是全局变量的地址也可能是局部变量的地址
void storeValue(string name, string regStr)
//生产mips代码并输出
void genMipsCode(mipsOpretion opp, string zz, string xx, string yy);
//关于数组的操作,lw和sw
void genMipsCode_array(mipsOpretion opp, string arrayName, string index, string
dstReg)
//将中间代码翻译成mips代码
void translate()
```

## 重点难点

**表达式部分的临时变量**，由于四元式只能有加减乘除的运算，所以一个表达式需要很多的临时变量，每进行一步运算都需要生成一个临时变量。关于表达式计算部分是递归下降的，我们用了传递指针的办法知道下一层的临时变量是什么：

```
int expression(string* opd);
int item(string* opd);
int factor(string* opd);
```

opd是上一层传递来的，代表着这一层的临时变量，当这一层的临时变量确定时（通过生成临时变量或者本身是常数），就需要修改opd指针，这样上一层就知道这一层所代表的临时变量是什么了。

**函数的目标代码生成**，函数生成是一个相对难的部分，需要保存环境和恢复环境。

首先让fp下沉到main函数的底部。

在调用函数之前，保存运行环境：

- $sp寄存器的地址
- 参数，参数被当做局部变量
- 参数以外的局部变量。
- 需要保存的寄存器
- $sp
- $ra
- $fp寄存器的地址

从函数返回后，需要恢复环境。

## 总结

总得来说，我认为代码生成部分还是比较难的。一方面，我们难以实现一个好的架构，在开始做代码生成的时候我甚至有点不知道要去做什么。另一方面，优化带来了诸多麻烦，由于开始没有想清楚，在优化的过程中我遇到了很多问题。

一学期的编译课程到这里就快结束了，我从中收获了很多，学到了很多新的东西，培养了自学能力和查阅资料的能力

# 代码优化阶段

## 常量优化

const类型直接转换为值。

```
const a=3,b=2;
c = a +b => c = 3 + 2
```

在进行运算时，常数的运算直接算出来，不通过mips运算，比如：

```
a=3+8 =>a=11
```

## 临时变量优化

### 优化前

所有局部变量（包括参数和临时变量）全部存在内存中。

每当需要使用时都从内存中取值。

每当需要赋值时都把值写到内存。

### 优化后

注意到临时变量都是临时的，它们都满足这样一个条件：

只有一次声明，只有一次使用。

观察到这个规律后，我们给他们分配t寄存器，在声明的时候分配寄存器，在使用的使用释放寄存器。

需要的数据结构：

```
bool tRegIsBusy[10] = { false, };//判断寄存器是否代表了某个临时变量
string tRegContent[10];//t寄存器中存的内容
```

```
//查找空闲的t寄存器
int findEmptyTReg()
//查找中间变量tempName是否在某个t某个寄存器中，如果找到反回寄存器的编号，否则返回-1
int findTempInReg(string tempName)
```

具体实现：

在声明临时变量时，查找有无空闲寄存器，如果有，则分配空闲寄存器，同时让记录空闲寄存器的值。

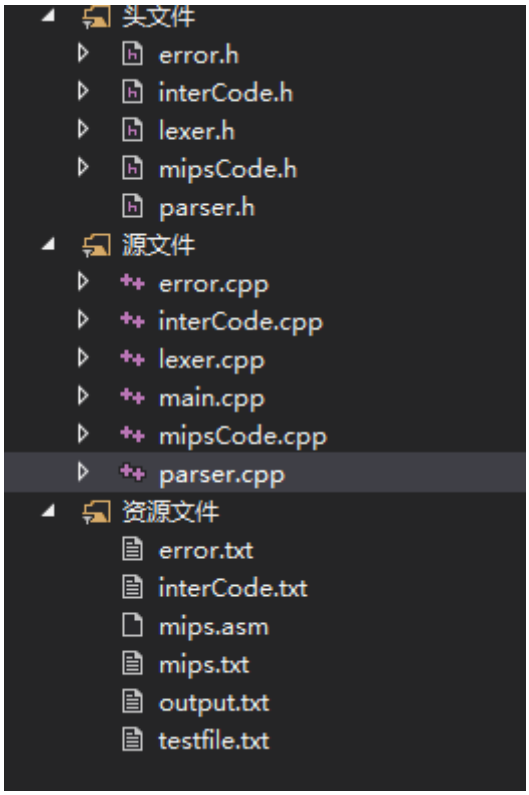
在使用临时变量时，释放临时寄存器，同时让寄存器中的内容清空。

## 总结

由于时间关系，我没有采取太多的优化，只是做了比较简单的，这对于我来说，可能是一种相对合理的分配。

# 最终总结

## 程序框架




## 感想

一学期下来，我总算完成了这个编译器，总的来说，这是一个不错的体验。

有时候我会觉得编译很难，可能需要花好几天的时间去完成一次作业；有时候我觉得编译很简单，甚至能将原理课上学到的知识融会贯通是一种乐趣。

这门课程让我学会了很多，我通过自学学会了一门语言，我通过学习他人经验使得自己走的弯路变少了，我能够将课上学到的知识运用，这种体验是很不错的。

希望我能变的更好，也希望课程组能做的更好。