《编译技术》课程设计文 档

学号：\_\_\_\_15061106\_\_\_\_\_\_\_\_

姓名：\_\_\_\_林 鑫\_\_\_\_\_\_\_\_\_

2018年 1月 14日

## 一．需求说明

### 1．文法说明

获取的文法为扩充C0文法（无实型），包含while循环。具体的文法如下：

＜加法运算符＞ ::= +｜-

＜乘法运算符＞ ::= \*｜/

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= ０｜＜非零数字＞

＜非零数字＞ ::= １｜．．．｜９

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

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

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

＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}

| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

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

＜常量＞ ::= ＜整数＞|＜字符＞

＜类型标识符＞ ::= int | char

＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’ //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数定义＞ ::= void＜标识符＞(’＜参数＞‘)’‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞‘}’//第一种选择为有参数的情况，第二种选择为无参数的情况

＜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＞}

＜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＞‘]’|‘(’＜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＞;|＜情况语句＞｜＜返回语句＞;

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

＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞else＜语句＞

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真

＜循环语句＞ ::= while ‘(’＜条件＞‘)’＜语句＞

＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞[＜缺省＞] ‘}’

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

＜情况子语句＞ ::= case＜常量＞：＜语句＞

＜缺省＞ ::= default : ＜语句＞

＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

＜语句列＞ ::= ｛＜语句＞｝

＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

＜写语句＞ ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’

＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

附加说明：

（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符不区分大小写字母

（3）写语句中的字符串原样输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

### 2．目标代码说明

生成的目标代码为MIPS汇编。

| 指令 | 含义 |
| --- | --- |
| ADD | 加 |
| SUB | 减 |
| MUL | 乘法 |
| DIV | 除法 |
| LI | 加载立即数 |
| LA | 取地址 |
| LW | 从存储器中读一个字的数据至寄存器中 |
| SW | 将寄存器中的值写入存储器中 |
| MOVE | 将一个寄存器中的值写入另一个寄存器 |
| SYSCALL | 系统调用 |
| JR | 跳转至某寄存器中存储的地址 |
| NOP | 空指令 |
| JAL | 跳转并链接 |
| J | 跳转 |
| BLE | 小于等于时跳转 |
| BGE | 大于等于时跳转 |
| BNE | 不等于时跳转 |
| BEQ | 等于时跳转 |
| BLT | 小于时跳转 |
| BGT | 大于时跳转 |

### 3. 优化方案\*

1. 基本块内公共子表达式删除

删除基本块内的公共子表达式，达到减少冗余计算指令的效果。

1. 全局寄存器分配

合理利用通用寄存器，提高程序的执行效率。

1. 常数合并

常数合并将能在编译时计算出值的表达式用其值替代，减少多余的计算和访存。

1. 窥孔优化（相反运算合并、赋值合并）

通过优化一个较短序列，如删除冗余指令、替换更高效的代码，来达到提升目标代码质量的目的。

本程序实现了相反运算合并和赋值合并两类优化。

## 二．详细设计

### 1．程序结构

本编译器的整个编译过程主要可分为两个步骤，第一步经过词法分析、语法分析、语法制导翻译模块将源程序翻译为中间代码（四元式形式），第二步在四元式的基础上经过优化，最终生成MIPS汇编目标码。程序具体结构如下图所示：

语法制导翻译

语法分析

词法分析

中间代码生成

代码优化

目标码生成

错误处理

符号表管理

### 2．类/方法/函数功能

| 返回类型 | 函数名 | 函数功能 |
| --- | --- | --- |
| 词法分析部分 | | |
| bool | isSpace(x), etc. | 一类宏函数，判断当前字符是否是各类字符（如空格、换行符、字母、数字、加号、引号等） |
| symbolSet | reserver() | 判断当前token是否为保留字 |
| symbolSet | otherChar(char x) | 判断当前无二义性单字符的类型 |
| const char\* | symbol\_type\_to\_str(type) | 根据当前单词枚举类型返回对应枚举名字符串 |
| void | error() | 分析过程中出现错误，跳过当前单词直至空白符或文件结束符 |
| void | getSym() | 分析下一个单词 |
|  |  |  |
| 语法分析和制导翻译部分 | | |
| void | syntax() | 语法分析总程序 |
| void | program() | 程序子程序 |
| void | constDec() | 常量声明子程序 |
| void | constDef() | 常量定义子程序 |
| int | numericDef() | 数值常量处理程序 |
| void | decHead() | 声明头部子程序 |
| void | varDec() | 变量声明子程序 |
| void | varDef() | 变量定义子程序 |
| void | funcDef() | 函数定义子程序 |
| void | paramList() | 参数子程序 |
| void | compound() | 复合语句子程序 |
| void | statementList() | 语句列子程序 |
| void | mainFunc() | 主函数子程序 |
| string | expr() | 表达式子程序 |
| string | term() | 项子程序 |
| string | factor() | 因子子程序 |
| void | statement() | 语句子程序 |
| void | ifState() | 条件语句子程序 |
| void | whileState() | 循环语句子程序 |
| string | funcCall() | 函数调用子程序 |
| void | assignState() | 赋值语句子程序 |
| void | readState() | 读语句子程序 |
| void | writeState() | 写语句子程序 |
| void | switchState() | 情况语句子程序 |
| vector | caseList() | 情况表子程序 |
| casestruct | caseSubState() | 情况子语句子程序 |
| string | caseDefault() | 情况缺省子程序 |
| void | returnState() | 返回语句子程序 |
| void | condition() | 条件子程序 |
| void | paramVal() | 值参数表子程序 |
|  |  |  |
| 符号表管理部分 | | |
| int | insertTable(name, kind, type, value, address, name) | 插入符号表函数 |
| void | updatePara(para) | 返填函数参数个数函数 |
| int | searchTable(name, func) | 查找符号表函数 |
| void | initTable() | 初始化函数 |
|  |  |  |
| 错误处理部分 | | |
| void | errmain(type, info) | 错误处理主程序 |
|  |  |  |
| 中间代码生成部分 | | |
| void | genQuaternion(opr, lvar, rvar, ret) | 生成四元式 |
| string | genNewLab() | 生成新的标签 |
| string | genNewVar() | 生成新的临时变量 |
| string | genStringLab() | 生成新的字符串常量标签 |
| void | dumpQuaternion() | 导出中间代码 |
|  |  |  |
| 目标代码生成部分 | | |
| void | genMipsCode() | 生成MIPS汇编代码 |
| int | addLocalVar(name, size) | 添加局部变量并返回其偏移地址 |
| int | findLocalVar(name) | 查找局部变量并返回其偏移地址 |
| void | initMips() | 初始化，填入数据段信息 |
| void | calc\_code(), etc. | 根据不同类别的四元式，生成对应代码 |
|  |  |  |
| 代码优化部分 | | |
| void | optimize() | 优化总函数 |
| void | copyBroadcast() | 复制传播 |
| void | DAGOptimize() | 删除公共子表达式 |
| void | constCombine() | 合并常数优化 |
| void | calcCombine() | 相反运算合并 |
| void | assignCombine() | 赋值合并 |
| void | refCount() | 全局寄存器分配 |

### 3．调用依赖关系

词法分析

语法分析和

制导翻译

中间代码生成

目标码生成

代码优化

符号表管理

错误处理

getSym()

isSpace(), etc.

otherChar()

reserve()

error()

errmain()

insertTable(), searchTable()

program()等递归子程序处理

genQuaternion()

genNewLab()

getMipsCode()

optimize()

DAGOptimize()

refCount()

constCombine()

主程序

### 4．符号表管理方案

符号表具体数据结构如下：

typedef struct

{ //符号表表项

string name; //标识符

symkind kind; //种类

symtype type; //变量类型或函数返回类型

int value; //常量值

int address; //存储地址或地址位移

int len; //数组长度或函数参数个数

} symTabItem;

typedef struct

{ //符号表

symTabItem item[TABSIZE];

int curpnt; //符号表当前位置指针

int totSub; //分程序总数

int subpnt[MAXLEV]; //分程序索引表

} symTable;

### 5．存储分配方案

1. 寄存器分配方案：计划采用引用计数法进行寄存器分配。

2. 运行栈结构：采用理论课程中介绍的活动记录，在运行时为每个分程序建立对应的活动记录，并在其退出时撤销该记录。活动记录中的内容如下：

局部数据区：存放模块中定义的各个局部变量。

形参数据区：存放函数的显式参数，每个参数都要分配数据空间，存放实参值或地址。

prev abp: 存放调用模块记录基地址。

ret addr: 存放返回地址，即调用语句的下一条执行指令地址。

ret value: 存放返回值，若无值为空。

display区：存放各外层模块活动记录的基地址。

参数区

AR2

|  |
| --- |
| 栈顶 |
| 局部数据区 |
| 形参数据区 |
| prev abp |
| ret addr |
| ret value |
| display区 |
| AR1 |

### 6. 解释执行程序\*

本程序无解释执行部分，编译生成MIPS汇编代码。

### 7. 四元式设计\*

本程序输出的中间代码为四元式，按照中缀表达式格式，即形如x = y op z，其中x为结果，y为左操作数，z为右操作数，op为操作符。为保证各类代码一致性，输出时采用的是“操作符，左操作数，右操作数，结果”的形式，与文档中要求稍有不同。

在程序中，每个四元式存储在如下结构中：

typedef struct

{ //四元式结构

oprSet opr; //操作符

string lvar, rvar, ret; //左操作数、右操作数和结果

} QCODE;

程序可能涉及的操作和对应四元式如下表所示：

|  |  |  |
| --- | --- | --- |
| 操作种类 | 四元式设计 | 说明 |
| const int one = 1, two = 2;  const char la = ‘a’; | const, int, 1, one  const, int, 2, two  const, char, 97, la | 常量声明 |
| int i;  int s[10];  char q, p; | var, int, , i  var, int, 10, s  var, char, , q  var, char, , p | 变量声明 |
| void quicksort(int l, int r) | func, void, , quicksort  para, int, , l  para, int, , r | 函数声明 |
| a = b \* (-c) + b\*( -c ); | -, 0, c, $\*t\_4  \*, b, $\*t\_4, $\*t\_5  -, 0, c, $\*t\_6  \*, b, $\*t\_6, $\*t\_7  +, $\*t\_5, $\*t\_7, $\*t\_8  =, $\*t\_8, , a | 表达式计算 |
| res = combinatorial(m, n); | parav, , , m  parav, , , n  call, combinatorial, , $\*t\_59  =, $\*t\_59, , res | 函数调用 |
| return (0); | ret, , , 0 | 函数返回 |
| i < 10 | <, i, 10, | 条件判断 |
|  | jne, , , .\_label\_33 | 不满足条件跳转 |
|  | jmp, , , .\_label\_35 | 无条件跳转 |
|  | lab, , , .\_label\_34 | 标签 |
| lower[0] = 97; | []=, 97, 0, lower | 为数组元素赋值 |
| i = lower[ind]; | =[], lower, ind, $\*t\_65  =, $\*t\_65, , i | 数组元素取值 |
| scanf(\_op);  scanf(m, n); | scanf, char, , \_op  scanf, int, , m  scanf, int, , n | 读语句 |
| printf(res);  printf("Not a letter: ", ch); | prntf, , res, int  prntf, .\_string\_5, ch, char | 写语句  (.\_string\_x)为内部标识 |

### 8. 目标代码生成方案\*

目标代码生成主要有以下任务：目标代码地址空间的划分、从中间代码到目标代码转换过程中的指令选择和寄存器分配指派的优化。

对全局寄存器分配，计划采用引用计数法，对变量进行有效的分配。

对于局部变量信息，程序使用如下数据结构记录：

typedef struct

{

string name;

int address;

} Localvar; //局部变量(标识符和地址)

vector<Localvar> localvar; //当前子程序的局部变量

在遇到局部变量时，将从向量数组中找到对应记录，并选择正确的偏移地址输出。向量数组在初始时和进入函数时将被清空。

目标代码生成时，首先生成的是数据段信息，包含全局变量、全局常量和字符串常量；之后开始生成代码段信息：从第一条四元式开始，根据每条四元式的内容生成对应的指令，直至程序结束。

**注：**由于本文法不包含换行符，为测试方便在生成目标代码时，程序处理为在每个输出语句后都有一个换行。此行为是被允许的。若要取消此功能，将mipscode.cpp中第311行代码删除或注释后，重新编译本编译器即可。

参考：http://judge.buaa.edu.cn/forum/thread.jsp?forum=451&thread=7421

### 9. 优化方案\*

基本块数据结构如下：

typedef struct

{

int id; //块id

string label; //块标签/函数名

vector<QCODE> midcodes; //基本快内的语句序列

vector<int> prev, next; //块的前驱和后继们

} basicBlock; //基本块

DAG图和节点表的数据结构如下：

struct DAGNODE

{

int id;

string name;

vector<int> parent;

int left, right;

};

typedef struct

{

string name;

int id;

bool init;

} DAGItem;

关键算法：

1. 基本块内部的公共子表达式删除
2. 全局寄存器分配（引用计数法）
3. 合并常数
4. 窥孔优化（相反运算合并、赋值合并）

### 10. 出错处理

当源代码在编译过程中遇到错误时，编译器会输出错误提示，包含出错位置所在行号和错误类别。编译器会尽可能地将错误影响最小化，跳过出错部分继续编译。

可能出现的错误如下：

|  |  |
| --- | --- |
| 错误类别 | 含义 |
| 词法分析部分 | |
| UNSIGNED\_DIGIT\_START\_WITH\_0 | 无符号整数出现前导0 |
| INVALID\_CHAR\_TYPE | 非法字符类型 |
| LACK\_OF\_QUOTE | 缺少单引号 |
| LACK\_OF\_DOUBLE\_QUOTE | 缺少双引号 |
| INVALID\_IDENTITY | 非法标识符 |
|  |  |
| 语法分析与制导翻译部分 | |
| SYMBOL TABLE FULL | 符号表满 |
| DUPLICATED\_DEFINE\_IDENTITY | 标识符多重定义 |
| UNDEFINED IDENTITY | 未定义的标识符或函数 |
| INVALID CONST TYPE | 常量声明类型错误 |
| CONST NOT ASSIGNED | 常量未赋值 |
| INVALID\_CONST\_VALUE | 常量值异常 |
| INVALID\_ARRAY\_LENGTH | 数组长度声明异常 |
| INVALID\_TYPE | 变量/常量类型错误 |
| UNEXPETED RETURNED VALUE | void型函数返回值 |
| LACK OF SEMICOLON | 缺少分号 |
| LACK OF RIGHT BRACKET | 缺少右中括号 |
| LACK OF LEFT PARENT | 缺少左小括号 |
| LACK OF RIGHT PARENT | 缺少右小括号 |
| LACK OF LEFT BRACE | 缺少左大括号 |
| LACK OF RIGHT BRACE | 缺少右大括号 |
| LACK\_OF\_COLON | 缺少冒号 |
| CONST ASSIGNMENT | 常数赋值 |
| MAIN\_TYPE\_ERROR | main函数不为void类型 |
| PARAMETER COUNT CHECK FAILED | 参数列表个数不匹配 |
| LACK\_OF\_CASE\_STATEMENT | 缺少case子语句 |
| UNEXPETED\_END\_OF\_FILE | 文件异常结束 |
| LACK\_OF\_ASSIGN\_MARK | 缺少赋值符号 |
| LACK\_OF\_ELSE\_STATEMENT | 缺少else语句段 |
| LACK\_OF\_MAIN\_FUNCTION | 缺少main函数或不在程序尾部 |
| ARRAY\_INDEX\_EXCEED | 数组下标访问越界 |

## 三．操作说明

### 1．运行环境

IDE环境：

Microsoft Visual Studio 2010

版本 10.0.40219.1 SP1Rel

已安装的版本: Ultimate

操作系统：

Microsoft Windows 10 专业版

1709 (OS内部版本16299.192)

### 2．操作步骤

1. 解压工程文件，得到工程目录compiler。
2. 打开工程目录，使用VS2010导入工程或直接双击解决方案compiler.sln来打开工程。
3. 在VS2010的“解决方案管理器”中选择“compiler”项目，点击菜单栏上的“生成”、“生成compiler”来编译编译器。
4. 输出栏将显示编译情况，当“编译成功”字样出现时表示编译器已生成。
5. 点击菜单栏上的“调试”、“开始执行(不调试)”或使用快捷键Ctrl+F5来运行编译器。
6. 编译程序执行，控制台提示如下：

Extend-C0 Compiler

Input File Path:

此时输入准备好的待编译程序路径（也可直接将文件拖到窗口内，路径会自动输入），按下回车。

1. 控制台显示输入的程序路径，并提示输入MIPS生成结果路径（可按下方向键“上”键来恢复上一次输入内容，并将后缀名改为asm，表示在相同目录下输出一个同名MIPS汇编文件）。确认输入无误后，按下回车。
2. 编译过程开始。若源程序有误，编译器会在控制台输出错误信息，包含行号和提示信息，并继续编译。
3. 编译结束时，控制台提示“MIPS result file generated!”，按任意键结束程序。编译器一共生成4个文件，分别为：中间代码（优化前）、中间代码（优化后）、目标代码（优化前）和目标代码（优化后）。中间代码与源程序同目录，文件名为源程序名+”.midcode”和源程序名+”.midcode.after”；目标代码（优化后）所在路径为MIPS生成结果路径（输入的）+”.after”。
4. 一种可能的操作情况如下：

控制台输出：

Extend-C0 Compiler

Input File Path: C:\Users\ILOFI\Documents\test\opt\_official.txt

Input File is: C:\Users\ILOFI\Documents\test\opt\_official.txt

MIPS result file path: C:\Users\ILOFI\Documents\test\opt\_official.asm

MIPS result file generated!

请按任意键继续. . .

此时各个文件的位置为：

中间代码（优化前）：C:\Users\ILOFI\Documents\test\opt\_official.txt.midcode

中间代码（优化后）：C:\Users\ILOFI\Documents\test\opt\_official.txt.midcode.after

目标代码（优化前）：C:\Users\ILOFI\Documents\test\opt\_official.asm

目标代码（优化后）：C:\Users\ILOFI\Documents\test\opt\_official.asm.after

## 四．测试报告

### 1．测试程序及测试结果

**测试程序一（正确）**

源代码（程序源代码较长，未放入文档中，详见“../测试程序/01.txt”）

测试结果（测试结果较长，未放入文档中，详见“../测试程序/01\_result.txt”）

**测试程序二（正确）**

源代码（程序源代码较长，未放入文档中，详见“../测试程序/02.txt”）

测试结果

测试用例1

input:

s

output:

All Sort Algorithm Result Accepted!

测试用例2

input:

c16

8

output:

12870

测试用例3

input:

n3

aZ-

output:

a Next letter: b

Z Next letter: a

- Not a letter: -

**测试程序三（正确）**

源代码

void main()

{

int a, b, c, d;

a = 3;

b = 1; c = -2;

d = 5;

printf("a=",a);

printf("b=",b);

printf("c=",c);

printf("d=", d);

d = a; a = b; b = c + c \* 2; c = d;

printf("a= ",a);

printf("b= ",b);

printf("c= ",c);

printf("d= ",d);

d = -c;

a = b \* (-c) + b\*( -c );

printf(a);

if (b\*c-a\*(-c)+d\*(-c)+b\*c < a)

printf("Hello world");

else

printf("Hello world2");

return;

}

测试结果

output:

a=3

b=1

c=-2

d=5

a= 1

b= -6

c= 3

d= 3

36

Hello world2

**测试程序四（正确）**

源代码

int a, b;

int fact(int u)

{

if (u == 0) return (1);

else return (u \* fact(u - 1));

}

int abs(int x)

{

if (x > 0) return (x);

else return (-x);

}

void main()

{

const int one = 1;

scanf(a, b);

printf(abs(a-b) \* fact(1) + 'a' - 'b' + 2 / 2 \* 3 / 3 \* one);

printf(fact(a)+fact(b));

}

测试结果

input:

7

10

output:

3

3633840

**测试程序五（正确）**

源代码

int min(int x, int y)

{

if (x < y) return (x);

else return (y);

}

void main()

{

const int five = 5;

int a[5], i, t;

int ans;

i = 0;

while (i < five)

{

scanf(t);

a[i] = t;

i = i + 1;

}

ans = a[1];

i = 1;

while (i < five)

{

ans = min(ans, a[i]);

i = i + 1;

}

printf(ans);

}

测试结果

input:

7

5

8

1

6

output:

1

**测试程序六（错误）**

源代码

const int ARRSIZE = 10, hehe;

const char f = '!', q = 't;

int a[010], arrsize;

void main()

{

int i, t, 9kkk;

printf("Hello world);

i = 0;

while (i < 10)

{

scanf(t);

a[i] = t;

i = i + 1;

fa = q;

}

i = 0;

while (i < 10)

{

printf(a[i]);

i = i + 1;

}

return;

}

报错信息

Error on line 1: CONST\_NOT\_ASSIGNED hehe

Error on line 2: INVALID\_CHAR\_TYPE !

Error on line 2: LACK\_OF\_QUOTE

Error on line 2: INVALID\_CONST\_VALUE t

Error on line 3: UNSIGNED\_DIGIT\_START\_WITH\_0

Error on line 3: DUPLICATED\_DEFINE\_IDENTITY arrsize

Error on line 7: INVALID\_IDENTITY

Error on line 8: LACK\_OF\_DOUBLE\_QUOTE Hello world);

Error on line 8: LACK\_OF\_RIGHT\_PARENT

Error on line 8: LACK\_OF\_SEMICOLON

Error on line 14: UNDEFINED\_IDENTITY fa

**测试程序七（错误）**

源代码

const char f = 500;

int a[-10];

void printa(charr b, inta c)

{

printf("a");

return (1)

}

void main()

{

int s[3;

}

报错信息

Error on line 1: INVALID\_CONST\_VALUE 500

Error on line 2: INVALID\_ARRAY\_LENGTH -

Error on line 4: INVALID\_TYPE charr

Error on line 4: INVALID\_TYPE inta

Error on line 7: UNEXPETED\_RETURN\_VALUE\_FUNCTION

Error on line 8: LACK\_OF\_SEMICOLON

Error on line 12: LACK\_OF\_RIGHT\_BRACKET

**测试程序八（错误）**

源代码

void test

printf("hello");

}

void main[)

{

int a, b;

a = 3; b = 4;

if (a > b

printf(a);

else ;

报错信息

Error on line 3: LACK\_OF\_LEFT\_BRACE

Error on line 6: LACK\_OF\_LEFT\_PARENT

Error on line 11: LACK\_OF\_RIGHT\_PARENT

Error on line 13: UNEXPETED\_END\_OF\_FILE

Error on line 13: LACK\_OF\_RIGHT\_BRACE

**测试程序九（错误）**

源代码

char \_op;

void hello{}

int main()

{

const int two = 1;

scanf(\_op);

switch (\_op)

{

default

{ printf("hello world!");}

}

two = 2;

}

报错信息

Error on line 5: MAIN\_TYPE\_ERROR

Error on line 11: LACK\_OF\_CASE\_STATEMENT

Error on line 12: LACK\_OF\_COLON

Error on line 14: CONST\_ASSIGNMENT two

**测试程序十（错误）**

源代码

int max(int x, int y)

{

if (x > y) return (x);

else return (y);

}

void test

{

int a[10];

a[0] 3;

if (a[0] > 5)

printf("Hello");

a[1] = max(3);

a[700] = 1;

}

报错信息

Error on line 10: LACK\_OF\_ASSIGN\_MARK

Error on line 13: LACK\_OF\_ELSE\_STATEMENT

Error on line 13: PARAMETER\_COUNT\_MISMATCH max

Error on line 14: ARRAY\_INDEX\_EXCEED 700

Error on line 15: LACK\_OF\_MAIN\_FUNCTION

### 2．测试结果分析

1. 测试一（正确）

本程序为综合测试程序，覆盖所有文法，重点考察输入输出语句、数组元素存取、函数递归调用以及各类语句。

程序模拟了一个简单的学生信息管理系统，通过与用户进行输入输出交互实现信息录入和查看。结果正确

1. 测试二（正确）

本程序为综合测试程序，覆盖所有文法，重点考察函数递归调用、数组元素存取、复杂表达式计算、变量命名域冲突、标识符大小写处理、输入输出语句等。

程序有三种功能，根据启动时输入的字符决定，分别为：排序测试（程序对同一组数据运行三个排序算法，并判断结果是否正确）、组合数计算和字符轮换。结果正确。

1. 测试三（正确）

本程序重点测试表达式计算，同时还测试了输入输出语句和条件语句等。

程序中一个片段会产生含有三个父节点的DAG图，另一个片段含有公共子表达式。结果正确。

1. 测试四（正确）

本程序重点测试函数递归调用、表达式计算和常数合并优化。

程序的功能是输入两个正整数，输出它们差的绝对值和阶乘和，通过查看四元式可以验证常数合并优化的正确性。结果正确。

1. 测试五（正确）

本程序重点测试输入输出、数组元素存取和函数调用。

程序功能是从输入的五个整数中，找出最小数，并输出。结果正确。

1. 测试六（错误）

本程序主要测试变量常量声明时易产生的错误和一部分语法语义错误。包含产量未赋值、非法字符类型、非法常量值、前导零整数等共11个错误。

第1行的常量hehe未赋值；第2行的常量字符叹号非法，根据文法要求常量不包含有叹号；同一行的常量t在赋值时缺少右单引号，是非法常量值；第3行的数组长度根据文法应为无符号整数，不能含有前导零，因此010报错；同一行变量arrsize与常量同名，报错（本文法不区分标识符大小写）；第7行的9kkk不是合法标识符（以数字开头）；第8行printf语句内的字符串缺少右双引号，故报错，程序将后续的右括号和分号也识别为字符串内容，故产生缺失错误；第15行变量fa未定义，属于未定义标识符，报错。

1. 测试七（错误）

本程序主要测试一部分语法和语义错误。包含非法常量值、非法数组长度、非法类型等共7个错误。

第1行对字符常量f所赋值不符合文法中的字符定义，属于非法常量值；第2行数组的大小设为-10，不符合无符号整数的定义，属于非法数组长度；第4行函数参数表中的charr和inta都属于非法变量类型；第7行，void函数返回了值，报错；同一行的return语句缺少分号；第12行的数组声明中缺少右中括号。

1. 测试八（错误）

本程序主要测试一部分常见的语法错误。包含左小括号缺失、异常文件的结束等共5个错误。

第3行，test函数缺少左大括号就进入语句，报错；第6行，main标识符后缺少小括号，报错；第10行，if语句的条件后缺少右小括号，报错；在第13行后程序结束，缺少了右大括号，也属于异常的文件结束。

1. 测试九（错误）

本程序主要测试常见的语法和语义错误。包含case缺失、为常量赋值等共4处错误。

第5行，main的返回值类型为int，与文法规定不符，属于非法的main类型；第11行，switch语句中只有default语句，根据文法说明，switch语句至少要有一个case子语句，故报错；第12行，default标识符后缺少冒号，报错；第14行，出现了为常量two赋值的情况，报错。

1. 测试十（错误）

本程序主要测试常见的语法和语义错误。包含else语句缺失、函数参数数目不符、数组下标越界等共5个错误。

第10行，为数组元素赋值时缺少了赋值符号，报错；第13行，条件语句没有else语句，不符合文法要求，报错；第13行，调用的max函数应有2个参数，而这里只传递1个，报错；第14行，数组下标取到了700，越界报错；整个程序缺少main函数，不符合文法规定，报错。

## 五．总结感想

时光飞逝，转眼间，两个月的编译技术课程设计即将进入尾声，我们也即将迎来课设最终考核。在这两个月的时间里，我从最简单的词法分析程序开始，到语法分析、中间代码生成、目标码生成，直到最后的代码优化，一步一步完成了一个简单的扩展C0文法编译器。

在完成设计的过程中，我更加深刻的理解了程序语言在计算机中的层次性，明白了从高级语言到汇编语言乃至可执行程序转换的过程，也就是编译的内涵。在最终设计中，生成的MIPS指令不过数十种，但这些指令的组合叠加却能表达出成千上万种功能不同的程序，这不禁让人感叹计算机编程语言的丰富表达能力和底层设计的简洁之美。

编译器的设计过程也锻炼了我工程化管理的能力，在设计过程中，我学着采用模块化思想来管理编译器，并使用版本控制工具记录我的代码变更流程。模块化的设计能使编译器具有更好的稳定性，也更容易测试和查错；版本控制工具除了能提供备份，也能清晰记录每个阶段的更改情况，方便后期盘查和回退。

当然，编译器的设计过程也不是一帆风顺的，常常会遇到困难，在发现问题和解决问题的过程中，我的分析能力、信息收集能力和处理问题的能力也得到了提升，相信这会给今后的学习研究带来较大的益处。

本次编译课设，我认为自己的时间分配和管理能力仍有待提升。尽管为了防止拖延现象的出现，老师为我们安排了每周一次的“检查点”，但到了后期，我仍然出现了时间紧张的现象，也导致了优化部分没有实现数据流分析和冲突图，是本次课设中较为遗憾的地方。在今后的学习生活中，我要加强自己的时间管理能力，学会将事情分段完成，给每件事安排一个适当的deadline，保证结果的质量。

最后，感谢老师和助教这一学期以来的帮助和支持！

注：【】内的文字为文档模板说明，完成的作业中需去掉。

标\*的章节需根据题目的难度进行取舍。