

****

编译技术课程设计

文 档

|  |  |
| --- | --- |
| 姓名： | 李娇阳 |
| 专业： | 计算机科学与技术 |
| 学院： | 北京学院 |
| 提交时间： | 2018年1月12日 |

## 一．需求说明

### 1．文法说明

我获得的文法是do-while循环的扩充C0文法。扩充C0文法较C文法简单一些，做了一些限制。它没有实型，只包含do-while循环。

### 2．目标代码说明

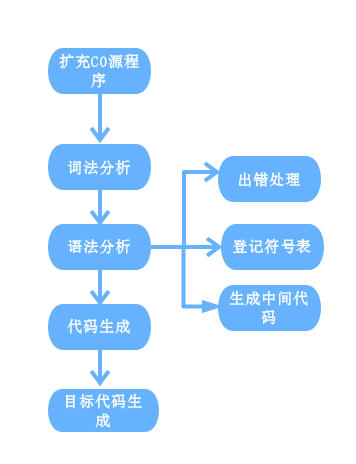
【说明要生成的目标代码指令及含义】

### 3. 优化方案\*

【说明需要完成的优化方案及其要求】

## 二．详细设计

### 1．程序结构



1. 在代码中，先读一遍源程序，生成词法分析结果。
2. 对词法分析结果进行语法分析。在语法分析的过程中进行错误检查和错误处理，并把变量、常量、函数名、临时变量等录入符号表，并记下他们所在的名字、类型、种类、层次、地址、函数的参数个数、常量的值。在语法分析过程中生成中间代码，并把中间代码写入结构体数组。
3. 读中间代码，生成目标代码。

### 2．类/方法/函数功能

|  |  |
| --- | --- |
| WordAnalysis——词法分析 | |
| void lineCheck(FILE \*stream) | 在输出类别码以前输出行号，当遇到\n以后lineflag设为0；如果lineflag是0，输出行号； |
| int isLetter(char a) | 判断大小写字母和下划线，若是返回1；否则返回0； |
| int isDigit(char a) | 判断数字0到9，若是返回数字；否则返回-1； |
| void ungetcha() | 数组下标回退一个单位； |
| char getcha(FILE \*stream) | 如果当前字符是tab键或回车键，数组下标自增1，返回-1；否则数组下标自增1，返回当前判断的符号；我认为这个很关键，在很多其他函数中都有对它的调用；如果这个写错了，整个程序都会错； |
| void Str\_CH(FILE \*stream,char c) | 用于输出字符串和字符；由于他们不做运算，便可直接输出； |
| int isSym(FILE \*stream,char c) | 根据运算符c返回相应的值一遍后期处理； |
| Int compare(int h) | 将当前标识符与保留字数组id内的保留字一一比较，若是保留字flag=1；否则flag=0； |
| void print(FILE \*stream,int k) | 输出保留字和标识符部分； |
| void printSym(FILE \*stream,int d) | 输出字符的类别码； |
| int WordAnalysis() | 生成词法分析结果，如果源程序文件打开失败，返回-1，否则不返回值； |

|  |  |
| --- | --- |
| Grammar——语法分析 | |
| int CopyIntoStruct() | 读词法分析结果写入结构体数组Str； |
| void nextsym() | 获取Str结构体数组内的下一个单词； |
| void Plus() | 加法运算符，按照字符和运算符分别考虑； |
| void Mult() | 乘法运算符，按照字符和运算符分别考虑； |
| void Rela() | 关系运算符； |
| void Alpha() | 判断字母，下划线也包含在内； |
| void Digital() | 判断数字，只能有1位，只能是0到9； |
| void Chara() | 判断字符； |
| void Strings() | 判断字符串，中间判断是否符合字符串要求； |
| void Program() | 分析程序； |
| void ConstantDescription() | 常量说明部分； |
| void ConstantDefinition() | 常量定义部分； |
| void UnsignedInteger() | 无符号整数定义； |
| void Integer() | 整数定义； |
| void Identifier() | 标识符定义； |
| void DeclareHead() | 声明头部； |
| void VariableDescription() | 变量说明； |
| void VariableDefinition() | 变量定义； |
| void TypeIdentifier() | 类型标识符； |
| void ReturnValueFunctionDefinition() | 有返回值函数定义； |
| void NoReturnValueFunctionDefinition() | 无返回值函数定义； |
| void CompoundStatement() | 复合语句； |
| void ParametersTable() | 参数表； |
| void MainFunction() | 主函数； |
| void Expression() | 表达式； |
| void Item() | 项； |
| void Factor() | 因子； |
| void Statement() | 语句； |
| void AssignmentStatement() | 赋值语句； |
| void ConditionalStatement() | 条件语句； |
| void Condition() | 条件； |
| void LoopStatement() | 循环语句； |
| void Constant() | 常量； |
| void SituationStatement() | 情况语句； |
| void SituationTable(char tmp[]) | 情况表； |
| void SituationSubSentence(char tmp[]) | 情况子语句；tmp表示switch语句的结尾，子语句执行完毕，跳到switch结尾； |
| void ReturnValueFunctionCallStatement() | 有返回值函数的调用语句； |
| void NoReturnValueFunctionCallStatement() | 无返回值函数的调用语句； |
| void ValueParameterTable(char fun[]) | 值参数表； |
| void StatementColumn() | 语句列； |
| void ReadSentences() | 读语句； |
| void WriteStatement() | 写语句； |
| void ReturnStatement() | 返回语句； |
| int GraFindTabType(int CurStrId) | 查找符号表，返回其类型；此函数只用在判断左右类型是否一致； |
| int GraFindTabObj(int CurStrId) | 查找符号表，返回参数种类，没找到返回-1；只用在查找是否定义过函数； |
| int findIdentT\_G(char name[]) | 从当前层次找，没找到再找全局；判断数组或标识符是否定义； |
| int findInTab(char name[]) | 判断左右类型是否一致； |

|  |  |
| --- | --- |
| ErrorHandling——错误处理 | |
| void Error(int e) | 定义错误类型，打印错误信息； |
| void Jump(int r) | 语法错误跳过，以便继续编译；跳过错误以后把错误标志设为零； |
| int findInTab(char name[]) | 判断左右类型是否一致； |

|  |  |
| --- | --- |
| KindsOfTabs——符号表管理 | |
| int enterTab(char enT\_name[],IdentType enT\_type,IdentObj enT\_obj,int enT\_value,int enT\_adr) | 登记符号表； |
| IdentType FindFuncName(char name[]) | 查找符号表返回函数类型； |
| int findDef(char name[]) | 查找符号表，判断是否定义过，未定义（没有重名）返回-1； |
| void findResult(int i\_findResult) | 如果找到同名就报错；用于检查全局变量常量以及函数名不能与任何元素同名； |
| void findTmpSame(int i\_findTmpSame) | 从当前层次开始找有没有重复定义；用于检测函数内常量变量有没有重复定义； |
| int findArrLen(int id) | 寻找数组长度，判断访问越界问题；找到返回长度，没找到返回-1 |
| void LableIntoMTab() | 为跳转添加标签，形如label\_1,label\_2； |
| void AddCaseLabel(int i) | 为switch的case语句添加标签； |
| void makeVar(char name[]) | 生成临时变量，形如tmp1,tmp2； |
| void makeEndLabel() | 生成end标签，形如end1518\_1； |
| void MakeSwitchEndLabel() | 生成switch的id，用于嵌套switch时的配对，形如switend\_0； |

|  |  |
| --- | --- |
| IntermediateCode——中间代码 | |
| void AddLabel(char Intername1[]) | 添加标签，形如label\_1； |
| void InsName1(Instr Interins,char Intername1[]) | 中间代码包含指令和名字一，形如Scan c，表示读入c； |
| void InsTypeName1(Instr Interins,CodeType Intertype,char Intername1[]) | 中间代码包含指令、类型、名字一，形如Gvar Int fbnum，表示声明一个int类型的全局变量fbnum； |
| void InsName1Res(Instr Interins,char Intername1[],char Interresult[]) | 中间代码包含指令、名字一、结果，形如Neg 1024 tmp0，表示把1024取负，存在结果tmp0里面； |
| void InsTypeName1Res(Instr Interins,CodeType Intertype,char Intername1[],char Interresult[]) | 中间代码包含指令、类型、名字一、结果，形如Gcon Int num1 1，表示声明int类型的全局常量num1，其值为1； |
| void InsName12Res(Instr Interins,char Intername1[],char Intername2[],char Interresult[]) | 中间代码包含指令、名字一、名字二、结果，形如Add letters 1 tmp3，表示把letters和1加起来，存在临时变量tmp3里面； |

|  |  |
| --- | --- |
| TargetCode——目标代码 | |
| void AddParaNumToFun() | 读符号表，给函数名那一行加上其参数个数； |
| void IntoRunStack() | 全局部分以及main函数内的常量变量存入结构体数组RunStack； |
| void CalInStack() | 读中间代码，生成目标代码； |
| void StoreData(int i\_StoreData,char Reg[5],LoadWhat S\_StoreData) | 把寄存器的值到内存中；i\_StoreData表示对应中间代码下标，Reg[5]表示要存哪个寄存器的值，S\_StoreData表示要存名字一、名字二、还是结果； |
| void LoadData(int i\_LoadData,char Reg[5],LoadWhat l\_LoadData,char name[]) | 从内存中取值到寄存器；i\_LoadData表示对应中间代码下标，Reg[5]表示要取哪个寄存器的值，l\_LoadData表示要取名字一、名字二、还是结果；name[]表示当前要分析的中间代码的name1/name2/result； |
| void DataDef(int i\_DataDef) | 全局数据段的定义，只定义全局常量； |
| void TmpDataDef(int i\_TmpDataDef,JudgeMain ifMain) | 函数内部临时常量定义；根据是否为main函数分情况讨论； |
| void StrDef() | 字符串数据段定义； |
| void AssDef(int i\_AssDef) | 赋值语句定义； |
| void RetDef(int i\_RetDef) | 有返回值的返回语句；   1. 取出返回值 2. 加载到v0 3. 跳转到函数的end标签 |
| void NRetDef(int i\_NRetDef) | 无返回值的返回语句；  直接跳转到函数的end标签； |
| void ReEndDef(int i\_EndDef) | 结束语句；   1. 生成end标签 2. 取出ra 3. sp加回来 4. jr $ra |
| void MainEndDef(int i\_MainEndDef) | main函数结束语句；  生成end标签； |
| void RetsDef(int i\_RetsDef) | 赋返回值语句；  把v0存进相应地址； |
| void ArrADef(int i\_ArrADef) | 向数组赋值定义；  分为全局数组和局部数组考虑； |
| void ArrGDef(int i\_ArrGDef) | 从数组取值定义；  分为全局数组和局部数组考虑； |
| int SwitDef(int i\_SwitDef) | 情况语句定义； |
| void PrinStrs(int i\_PrinStr) | 写语句； |
| void ScanIdent(int i\_ScanIdent) | 读语句； |
| void CompareOp(int i\_CompareOp) | 比较运算； |
| void Arithmetic(int i\_Arithmetic) | 加减乘除； |
| int CallParaFun(int i\_CallFun) | 调用函数定义； |
| void ParaIntoStack(int i\_ParaIntoStack) | 参数进入参数栈； |
| void ParaOutStack() | 参数离开参数栈； |
| void CallNoParaFun(int i\_CallNoParaFun) | 调用无参数函数定义；直接jal 函数名； |
| void FindStrLabel(char CurStr[]) | 输出字符串时寻找相应数据段的Str下标；用在开头声明字符串数据段的部分； |
| void SpliceStrId(char CurStr[]) | 根据Str的下标拼接出Str的标签； |
| int FindAdr(int id\_findadr,LoadWhat which,ReturnWhat Rwhat) | 在结构体数组RunStack中寻找全局的地址、类型、种类、或是否为char类型的标志； |
| int FindTmpIdentAdr(int i\_FindTmpIdentAdr,LoadWhat which,ReturnWhat rewhich) | 在符号表中查找临时常量、变量、数组的地址； |
| int FindParaAdr(int i\_FindParaAdr,int id\_Para) | 寻找参数对应的地址； |
| int FindFunLen(int i\_FindFunLen) | 寻找函数的长度，即其最大地址； |
| int FindParaIndex(int i\_FindParaIndex) | 转换参数的下标； |
| JudgeMain FindIdentLoc(int i\_FindIdentLoc,LoadWhat l\_FindIdentLoc) | 判断所在层次；   1. 如果是在main函数里面，返回JustInMain； 2. 如果是在非main函数里面，返回NotMain； 3. 如果是全局，返回IsMain； |
| int WhichCase(int i\_WhichCase) | 寻找应该跳转到哪个case； |

### 3．调用依赖关系

|  |
| --- |
| ErrorHandling——错误处理 |
| —— |

|  |  |
| --- | --- |
| Grammar——语法分析 | |
| int CopyIntoStruct() | —— |
| void nextsym() | —— |
| void Plus() | 需要nextsym()读取下一个单词； |
| void Mult() | 需要nextsym()读取下一个单词； |
| void Rela() | 需要nextsym()读取下一个单词； |
| void Alpha() | 需要isLetter()判断是否为字母或下划线；  需要nextsym()读取下一个单词； |
| void Digital() | 需要isDigit()判断是否为数字；  需要nextsym()读取下一个单词； |
| void Chara() | 分情况讨论，判断进入哪个函数；  Plus() Mult() Alpha() Digital(); |
| void Strings() | 不符合字符串要求，需要Error()；  需要nextsym()读取下一个单词； |
| void Program() | 分情况讨论，判断进入哪个函数；  ConstantDescription();  VariableDescription();  ReturnValueFunctionDefinition();  NoReturnValueFunctionDefinition();  MainFunction(); |
| void ConstantDescription() | 需要nextsym()读取下一个单词；  缺少分号时，需要Error()；  需要Jump()跳过错误代码段；  需要调用ConstantDefinition()； |
| void ConstantDefinition() | 需要nextsym()读取下一个单词；  需要Identifier()；  需要Jump()跳过错误代码段；  需要findResult()判断是否有重名全局常量；  需要findTmpSame()判断是否有在函数内有重名；  需要Integer()；  需要enterTab()录入符号表；  需要InsTypeName1Res()写入四元式；  需要Error();  需要Chara(); |
| void UnsignedInteger() | 需要isDigit()判断是否为数字；  需要Error();  需要nextsym()读取下一个单词； |
| void Integer() | 需要Plus()；  需要UnsignedInteger()；  需要makeVar()生成临时变量；  需要enterTab()录入符号表；  需要InsName1Res()写入四元式； |
| void Identifier() | 需要isLetter()判断是否为字母或下划线；  需要isDigit()判断是否为数字；  需要Error();  需要nextsym()读取下一个单词； |
| void DeclareHead() | 需要nextsym()读取下一个单词；  需要Identifier()；  需要findResult()判断是否有重名函数名；  需要enterTab()录入符号表；  需要InsTypeName1()写入四元式；  需要Error(); |
| void VariableDescription() | 需要VariableDefinition()；  需要nextsym()读取下一个单词；  需要Error();  需要Jump()跳过错误代码段； |
| void VariableDefinition() | 需要TypeIdentifier()；  需要Jump()跳过错误代码段；  需要Identifier()；  需要findResult()判断是否有重名全局变量；  需要findTmpSame()判断是否有在函数内有重名；  需要enterTab()录入符号表；  需要InsTypeName1()写入四元式；  需要nextsym()读取下一个单词；  需要UnsignedInteger()；  需要Error();  需要InsTypeName1Res()写入四元式； |
| void TypeIdentifier() | 需要nextsym()读取下一个单词；  需要Error(); |
| void ReturnValueFunctionDefinition() | 需要DeclareHead()；  需要nextsym()读取下一个单词；  需要ParametersTable()；  需要CompoundStatement()；  需要Error();  需要makeEndLabel();  需要InsName1Res()写入四元式；  需要enterTab()录入符号表；  需要Jump()跳过错误代码段； |
| void NoReturnValueFunctionDefinition() | 需要nextsym()读取下一个单词；  需要Identifier()；  需要InsTypeName1()写入四元式；  需要findResult()判断是否有重名函数名；  需要enterTab()录入符号表；  需要ParametersTable()；  需要CompoundStatement()；  需要makeEndLabel();  需要InsName1Res()写入四元式；  需要Error(); |
| void CompoundStatement() | 需要ConstantDefinition()；  需要VariableDescription()；  需要StatementColumn()； |
| void ParametersTable() | 需要TypeIdentifier()；  需要Identifier()；  需要Jump()跳过错误代码段；  需要enterTab()录入符号表；  需要InsTypeName1()写入四元式；  需要nextsym()读取下一个单词； |
| void MainFunction() | 需要nextsym()读取下一个单词；  需要enterTab()录入符号表；  需要InsTypeName1()写入四元式；  需要CompoundStatement()；  需要makeEndLabel();  需要InsName1()写入四元式；  需要Error(); |
| void Expression() | 需要Plus();  需要Item();  需要makeVar();  需要enterTab()录入符号表；  需要InsName1Res()写入四元式；  需要InsName12Res()写入四元式； |
| void Item() | 需要Factor()；  需要nextsym()读取下一个单词；  需要makeVar();  需要enterTab()录入符号表；  需要InsName12Res()写入四元式； |
| void Factor() | 需要ReturnValueFunctionCallStatement()；  需要Identifier()；  需要GraFindTabType();  需要findIdentT\_G();  需要nextsym()读取下一个单词；  需要Expression()；  需要makeVar();  需要enterTab()录入符号表；  需要InsName12Res()写入四元式；  需要Error();  需要Integer()；  需要Chara()； |
| void Statement() | 需要nextsym()读取下一个单词；  需要ConditionalStatement()；  需要LoopStatement()；  需要SituationStatement()；  需要StatementColumn()；  需要Error();  需要FindFuncName();  需要ReturnValueFunctionCallStatement()；  需要NoReturnValueFunctionCallStatement()；  需要Jump()跳过错误代码段；  需要AssignmentStatement()；  需要ReadSentences()；  需要WriteStatement()； |
| void AssignmentStatement() | 需要Identifier()；  需要GraFindTabType();  需要findArrLen();  需要nextsym()读取下一个单词；  需要Expression()；  需要findInTab();  需要InsName1Res()写入四元式；  需要Error();  需要InsName12Res()写入四元式；  需要Jump()跳过错误代码段； |
| void ConditionalStatement() | 需要nextsym()读取下一个单词；  需要Condition()；  需要InsName1Res()写入四元式；  需要AddLabel();  需要Error();  需要Jump()跳过错误代码段； |
| void Condition() | 需要Expression()；  需要Rela();  需要makeVar();  需要enterTab()录入符号表；  需要InsName12Res()写入四元式； |
| void LoopStatement() | 需要LableIntoMTab()；  需要AddLabel();  需要nextsym()读取下一个单词；  需要Statement()；  需要Condition()；  需要InsName1Res()写入四元式；  需要nextsym()读取下一个单词；  需要Error(); |
| void Constant() | 需要Integer()；  需要Chara()；  需要isDigit()判断是否为数字；  需要Error(); |
| void SituationStatement() | 需要MakeSwitchEndLabel()；  需要nextsym()读取下一个单词；  需要Expression()；  需要InsName1Res()写入四元式；  需要SituationTable(char tmp[])；  需要InsName1()写入四元式；  需要InsTypeName1Res()写入四元式；  需要Error(); |
| void SituationTable(char tmp[]) | 需要SituationSubSentence(char tmp[])；  需要Error();  需要Jump()跳过错误代码段； |
| void SituationSubSentence(char tmp[]) | 需要nextsym()读取下一个单词；  需要Constant()；  需要AddCaseLabel(int i)；  需要Statement()；  需要InsName1()写入四元式；  需要Error(); |
| void ReturnValueFunctionCallStatement() | 需要GraFindTabObj(int CurStrId)；  需要Identifier()；  需要Error();  需要Jump()跳过错误代码段；  需要GraFindTabType(int CurStrId)；  需要InsName1()写入四元式；  需要nextsym()读取下一个单词；  需要ValueParameterTable(char fun[])；  需要makeVar()生成临时变量；  需要enterTab()录入符号表；  需要InsTypeName1()写入四元式； |
| void NoReturnValueFunctionCallStatement() | 需要Identifier()；  需要Jump()跳过错误代码段；  需要InsName1()写入四元式；  需要nextsym()读取下一个单词；  需要ValueParameterTable(char fun[])；  需要Error(); |
| void ValueParameterTable(char fun[]) | 需要Expression()；  需要InsName12Res()写入四元式；  需要nextsym()读取下一个单词； |
| void StatementColumn() | 需要Statement()； |
| void ReadSentences() | 需要nextsym()读取下一个单词；  需要Identifier()；  需要InsName1()写入四元式；  需要Error(); |
| void WriteStatement() | 需要nextsym()读取下一个单词；  需要Strings()；  需要Expression()；  需要InsName1()写入四元式；  需要Error(); |
| void ReturnStatement() | 需要nextsym()读取下一个单词；  需要Expression()；  需要Error();  需要InsName1()写入四元式； |
| int GraFindTabType(int CurStrId) | —— |
| int GraFindTabObj(int CurStrId) | —— |
| int findIdentT\_G(char name[]) | —— |
| int findInTab(char name[]) | —— |

|  |  |
| --- | --- |
| IntermediateCode——中间代码 | |
| void AddLabel(char Intername1[]) | —— |
| void InsName1(Instr Interins,char Intername1[]) | —— |
| void InsTypeName1(Instr Interins,CodeType Intertype,char Intername1[]) | —— |
| void InsName1Res(Instr Interins,char Intername1[],char Interresult[]) | —— |
| void InsTypeName1Res(Instr Interins,CodeType Intertype,char Intername1[],char Interresult[]) | —— |
| void InsName12Res(Instr Interins,char Intername1[],char Intername2[],char Interresult[]) | —— |

|  |  |
| --- | --- |
| KindsOfTabs——符号表管理 | |
| int enterTab(char enT\_name[],IdentType enT\_type,IdentObj enT\_obj,int enT\_value,int enT\_adr) | —— |
| IdentType FindFuncName(char name[]) | —— |
| int findDef(char name[]) | —— |
| void findResult(int i\_findResult) | 需要findDef(); |
| void findTmpSame(int i\_findTmpSame) | —— |
| int findArrLen(int id) | —— |
| void LableIntoMTab() | —— |
| void AddCaseLabel(int i) | 需要InsTypeName1Res()写入四元式； |
| void makeVar(char name[]) | —— |
| void makeEndLabel() | —— |
| void MakeSwitchEndLabel() | —— |

|  |  |
| --- | --- |
| TargetCode——目标代码 | |
| void AddParaNumToFun() | —— |
| void IntoRunStack() | —— |
| void CalInStack() | 需要DataDef()；  需要TmpDataDef();  需要CompareOp();  需要ParaIntoStack();  需要CallParaFun();  需要CallNoParaFun();  需要PrinStrs();  需要ScanIdent();  需要Arithmetic();  需要AssDef();  需要RetDef();  需要NRetDef();  需要ReEndDef();  需要RetsDef();  需要ArrADef();  需要ArrGDef();  需要SwitDef();  需要MainEndDef()； |
| void StoreData(int i\_StoreData,char Reg[5],LoadWhat S\_StoreData) | 需要FindIdentLoc();  需要FindTmpIdentAdr();  需要FindAdr(); |
| void LoadData(int i\_LoadData,char Reg[5],LoadWhat l\_LoadData,char name[]) | 需要isDigit()判断是否为数字；  需要FindIdentLoc();  需要FindTmpIdentAdr();  需要FindAdr(); |
| void DataDef(int i\_DataDef) | 需要FindAdr(); |
| void TmpDataDef(int i\_TmpDataDef,JudgeMain ifMain) | 需要FindTmpIdentAdr();  需要FindAdr(); |
| void StrDef() | —— |
| void AssDef(int i\_AssDef) | 需要LoadData();  需要StoreData(); |
| void RetDef(int i\_RetDef) | 需要LoadData(); |
| void NRetDef(int i\_NRetDef) | —— |
| void ReEndDef(int i\_EndDef) | —— |
| void MainEndDef(int i\_MainEndDef) | —— |
| void RetsDef(int i\_RetsDef) | 需要StoreData(); |
| void ArrADef(int i\_ArrADef) | 需要FindIdentLoc();  需要FindTmpIdentAdr();  需要LoadData();  需要FindAdr(); |
| void ArrGDef(int i\_ArrGDef) | 需要FindIdentLoc();  需要FindTmpIdentAdr();  需要LoadData();  需要StoreData();  需要FindAdr(); |
| int SwitDef(int i\_SwitDef) | 需要LoadData();  需要WhichCase(); |
| void PrinStrs(int i\_PrinStr) | 需要SpliceStrId();  需要isDigit()判断是否为数字；  需要LoadData();  需要FindIdentLoc();  需要FindTmpIdentAdr();  需要FindAdr(); |
| void ScanIdent(int i\_ScanIdent) | 需要FindIdentLoc();  需要FindTmpIdentAdr();  需要FindAdr();  需要StoreData(); |
| void CompareOp(int i\_CompareOp) | 需要LoadData(); |
| void Arithmetic(int i\_Arithmetic) | 需要LoadData(); |
| int CallParaFun(int i\_CallFun) | 需要ParaIntoStack();  需要LoadData();  需要FindFunLen();  需要FindParaAdr();  需要FindParaIndex();  需要ParaOutStack(); |
| void ParaIntoStack(int i\_ParaIntoStack) | —— |
| void ParaOutStack() | —— |
| void CallNoParaFun(int i\_CallNoParaFun) | —— |
| void FindStrLabel(char CurStr[]) | —— |
| void SpliceStrId(char CurStr[]) | 需要FindStrLabel(); |
| int FindAdr(int id\_findadr,LoadWhat which,ReturnWhat Rwhat) | —— |
| int FindTmpIdentAdr(int i\_FindTmpIdentAdr,LoadWhat which,ReturnWhat rewhich) | —— |
| int FindParaAdr(int i\_FindParaAdr,int id\_Para) | —— |
| int FindFunLen(int i\_FindFunLen) | —— |
| int FindParaIndex(int i\_FindParaIndex) | —— |
| JudgeMain FindIdentLoc(int i\_FindIdentLoc,LoadWhat l\_FindIdentLoc) | —— |
| int WhichCase(int i\_WhichCase) | —— |

|  |  |
| --- | --- |
| WordAnalysis——词法分析 | |
| void lineCheck(FILE \*stream) | Str\_CH; printSym; Error; main; |
| int isLetter(char a) | 在Error; main;函数中调用； |
| int isDigit(char a) | 在Error; main;函数中调用； |
| void ungetcha() | 在isSym;main;函数中调用； |
| char getcha(FILE \*stream) | 在Str\_CH;isSym;Error; main;函数中调用； |
| void Str\_CH(FILE \*stream,char c) | 在WordAnalysis;函数中调用； |
| int isSym(FILE \*stream,char c) | 在WordAnalysis;函数中调用； |
| Int compare(int h) | 在WordAnalysis;函数中调用; |
| void print(FILE \*stream,int k) | 在WordAnalysis;函数中调用； |
| void printSym(FILE \*stream,int d) | 在WordAnalysis;函数中调用； |
| int WordAnalysis() | 在main;函数中调用； |

### 4．符号表管理方案

|  |  |
| --- | --- |
| 标识符类型IdentType | |
| VOID | 0 |
| INT | 1 |
| CHAR | 2 |

|  |  |  |
| --- | --- | --- |
| 标识符种类IdentObj | | |
| PARA | 参数 | 0 |
| FUNC | 函数名 | 1 |
| GVAR | 全局变量 | 2 |
| GCON | 全局常量 | 3 |
| GARR | 全局数组 | 4 |
| TVAR | 临时变量 | 5 |
| TCON | 临时常量 | 6 |
| TARR | 临时数组 | 7 |
| LABEL | 标签 | 8 |
| GTVAR | 全局临时变量 | 9 |
| TTVAR | 局部临时变量 | 10 |
| NOREEND | 无返回值的函数结尾 | 11 |
| REEND | 有返回值的函数结尾 | 12 |
| MAINEND | main函数结尾 | 13 |

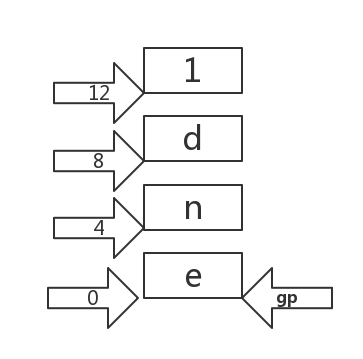
|  |
| --- |
| 符号表内容 |
| 标识符名字 name[100] |
| 标识符类型 type |
| 标识符种类 obj |
| 标识符地址 adrFirst |
| 维数 lev |
| 参数个数AllParaNum |

管理：在声明变量、常量、以及临时变量的时候，把它们录入符号表，赋予相应地址。以便后期运算时，存到相应内存。

### 5．存储分配方案

|  |
| --- |
| 字符串 |
| 存在数据段(.data) |

|  |
| --- |
| 全局以及main函数的变量常量 |
| 存在以gp为基地址的内存中 |



例如：

Gcon Char ch1 'e'

Gcon Char ch2 'n'

Gcon Char ch3 'd'

Gcon Int num1 1

ch1 的地址是0

ch2 的地址是4

ch3 的地址是8

num1 的地址是12

|  |
| --- |
| 非main函数的变量常量 |
| 存在以sp为基地址的内存中，运行结束后释放其空间 |

例如：

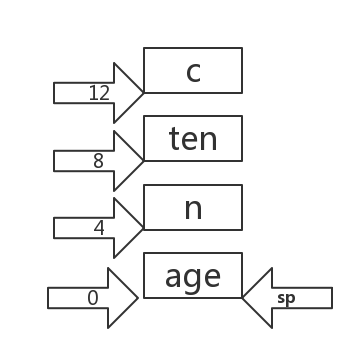
int age(int n){

const int ten = 10;

int c;

}

|  |  |
| --- | --- |
| age函数的变量常量 | 地址 |
| age | 0 |
| n | 4 |
| ten | 8 |
| c | 12 |



[注]：当跳转到函数时，让sp减去函数的最高地址，此处是12，然后在以sp为基地址的空间上对函数进行运算。当函数运行结束，再把sp加回来，释放空间。

**6. 四元式设计\***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 四元式 | | | | | |
| 指令 | | 类型 | 名字一 | 名字二 | 结果 |
| Gcon | 全局常量 | Char | Ch1 |  | ‘e’ |
| Gvar | 全局变量 | Int | fbnum |  |  |
| Garr | 全局数组 | Int | chb |  | 6 |
| Func | 函数名 | Int | f |  |  |
| Para | 参数名 | Int | x |  |  |
| Tcon | 临时常量 | Int | z |  | 1 |
| Tvar | 临时变量 | Int | i |  |  |
| Tarr | 临时数组 | Int | stpos |  | 100 |
| Grea | 大于 |  | x | ‘c’ | tmp48 |
| GaE | 大于等于 |  | key | 6 | tmp72 |
| Less | 小于 |  | key | 4 | tmp48 |
| LaE | 小于等于 |  | key | 4 | tmp48 |
| NaE | 不等于 |  | key | 4 | tmp48 |
| Equ | 等于 |  | key | 4 | tmp90 |
| Goto | 直接跳转 |  | label\_14 |  |  |
| TGoto | 为正跳转 |  | tmp4 |  | label\_1 |
| FGoto | 为负跳转 |  | tmp99 |  | label\_13 |
| Ret | 返回语句 |  | num |  |  |
| NRet | 无返回语句 |  | NullStatement |  |  |
| RetS | 返回值 | Char | tmp105 |  |  |
| Add | 加 |  | tmp101 | tmp102 | tmp103 |
| Min | 减 |  | a | tmp81 | tmp10 |
| Mul | 乘 |  | a | b | tmp65 |
| Div | 除 |  | a | b | tmp73 |
| Neg | 取反 |  | a |  | tmp91 |
| Ass | 赋值 |  | a |  | tmp93 |
| CallBegin | 调用开始 |  | f |  |  |
| CallEnd | 调用结束 |  | f |  |  |
| Push | 参数入栈 |  | tmp105 | PARA1 | f |
| End | 函数结尾 |  | end1518\_5 |  | nest |
| ArrA | 数组赋值 |  | b | tmp44 | 2333 |
| ArrG | 数组取值 |  | a | tmp43 | tmp44 |
| Scan | 读语句 |  | cho |  |  |
| Prin | 写语句 |  | d |  |  |
| Swit | 情况语句 |  | e |  | switend\_2 |
| SwitEnd | 情况语句结尾 |  | TheEndOfSwitch |  |  |
| ThisLabel | 添加标签 | Label | labelcon\_9 |  | ‘b’ |

[注]：上表为所有出现的四元式指令种类，表中的值代表该指令的例子。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 四元式 | | | | |
| 指令 |  | 名字一 |  |  |
| 指令 | 类型 | 名字一 |  |  |
| 指令 |  | 名字一 |  | 结果 |
| 指令 | 类型 | 名字一 |  | 结果 |
| 指令 |  | 名字一 | 名字二 | 结果 |

[注]:上表为所有四元式的类型。

### 8. 目标代码生成方案\*

|  |
| --- |
| 例如：函数Fun对应的mips汇编代码  .text  j main #首先统一函数头部跳转到main函数  Fun:  addu $sp $sp -36 #假设函数Fun的最高地址是36  sw $ra 36($sp) #把ra存到以sp为基地址的函数最高地址对应内存处 |

|  |
| --- |
| #sw的相关代码   1. 根据参数可知需要存name1还是name2还是result 2. 判断要存的数是全局还是main函数内还是非main函数内 3. 若为非main函数，例如：sw $t0 4($sp) 表示这个数在非main函数的地址是4，以sp为基地址来存 4. 若为main函数或在全局范围内，例如：sw $t0 32($gp) 表示这个数的地址是32，以gp为基地址来存 |

|  |
| --- |
| #lw的相关代码   1. 如果是整数或是字符，li $t0 3 2. 判断要取的数是全局还是main函数内还是非main函数内 3. 若为非main函数，例如：lw $t0 4($sp) 4. 若为main函数或在全局范围内，例如：lw $t0 32($gp) |

|  |
| --- |
| #全局常量的定义   1. 用li $t0 3 这种形式先把要存的数放在寄存器t0 2. 求该常量的地址addr，sw $t0 addr($gp) |

|  |
| --- |
| #临时常量定义   1. 用li $t0 3 这种形式先把要存的数放在寄存器t0 2. 若为非main函数，例如：lw $t0 4($sp) 3. 若为main函数或在全局范围内，例如：lw $t0 32($gp) |

|  |
| --- |
| #字符串定义  在mips汇编代码开头把所有字符串罗列出来  形如：  .data  Str0: .asciiz ""  Str1: .asciiz " !#$%&'()\*+,-./"  Str2: .asciiz "0123456789:;<=>?@"  Str3: .asciiz "ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`"  Str4: .asciiz "abcdefghijklmnopqrstuvwxyz{|}~"  LineBreaks: .asciiz "\n" |

|  |
| --- |
| #赋值语句   1. lw result到$t0 2. sw $t0 到name1 |

|  |
| --- |
| #有返回值的返回语句   1. lw name1 到$t0 2. addi $v0 $t0 0 #把t0加载到v0 3. 跳转到函数的end标签，例如：j end1518\_2 |

|  |
| --- |
| #非main函数结束语句   1. 添加结束标签，例如end1518\_2: 2. lw $ra 36($sp) 取出下一条指令的位置 3. addu $sp $sp 36 sp加回来 4. jr $ra |

|  |
| --- |
| #main函数结束语句  输出end1518\_5: |

|  |
| --- |
| #赋返回值  sw $v0 到name1 |

|  |
| --- |
| #向数组赋值   1. 首先判断数组是全局还是main函数内的数组还是非main函数内的数组 2. 如果是非main函数内：   1）la $t0 name1的地址 #数组名的地址在t0  2）lw name2到$t1 #数组下标在t1  3）lw result 到$t2 #要存入数组的值在t2  4）t1 乘以4  5）t0 = t0 + t1 #该下标的地址  6）addu $sp $sp $t0 让sp加上对应下标的地址  7）sw $t2 0($sp)  8）subu $sp $sp $t0 sp的偏移再减回来  3.main函数内定义的数组或是全局数组同第2条，只是sp换成gp |

|  |
| --- |
| #情况语句定义   1. 取出表达式的值存到t0 2. case语句后的常量以及各自的标签录入SwitCase结构体数组 |

|  |
| --- |
| #写语句  分字符串、整数、字符、标识符 四种情况讨论 |

|  |
| --- |
| #读语句   1. 判断标识符是全局还是main函数内的还是非main函数内的 2. 如果是整数：li $v0 5 3. 如果是字符：li $v0 12 4. 输出syscall 5. sw name1 到v0 |

|  |
| --- |
| #比较运算   1. lw name1 到t0 2. lw name2 到t1 3. 大于：slt $t2 $t1 $t0 4. 大于等于：sge $t2 $t0 $t1 5. 小于：slt $t2 $t0 $t1 6. 小于等于：sle $t2 $t0 $t1 7. 不等于：sne $t2 $t0 $t1 8. 等于：seq $t2 $t0 $t1 9. 若满足条件，t2值为1 |

|  |
| --- |
| #加减乘除   1. lw name1 到 t0 2. lw name2 到 t1 3. 加减乘除、取负 4. sw $t2 到result |

|  |
| --- |
| #调用函数定义   1. 在遇到第一个callend之前，所有参数进栈 2. 遇到callend以后，直到遇到callbegin之前，lw栈内的参数到t0 3. sp减去调用的函数最高地址 4. sw t0 到相应地址 5. 再把sp加回来 6. 该参数离开参数栈 |

|  |
| --- |
| #读到callend  jal 对应函数 |

### 9. 优化方案\*

在生成目标代码的时候，我没有进行全局寄存器分配，而是选择了几个固定的寄存器，来进行临时存数，随机把他们存入内存中。只作了下面几个小优化，稍稍提升一些效率。

|  |  |  |
| --- | --- | --- |
| 中间代码 | 含义 | 优化处理 |
| Gvar | 全局变量声明 | 因为全部存在内存，不在数据段声明 |
| Garr | 全局数组声明 | 因为全部存在内存，不在数据段声明 |
| Tvar | 函数内变量声明 | 存在函数对应的内存，不需声明，直接使用其地址存取 |
| Tarr | 函数内数组声明 | 存在函数对应的内存，不需声明，直接使用其地址存取 |
| Para | 参数声明 | 存在函数对应的内存，不需声明，直接使用其地址存取 |

### 出错处理

|  |  |  |
| --- | --- | --- |
| 错误序号 | 错误含义 | 错误解释 |
| 1 | lack ( L\_S\_Par | 缺少左小括号 |
| 2 | lack ) R\_S\_Par | 缺少右小括号 |
| 3 | lack ] R\_M\_Par | 缺少右中括号 |
| 4 | lack { L\_B\_Par | 缺少左大括号 |
| 5 | lack } R\_B\_Par | 缺少右大括号 |
| 6 | lack ; Semi | 缺少分号 |
| 7 | lack = Assign | 缺少等号 |
| 8 | lack : Colon | 缺少冒号 |
| 9 | lack main | 缺少main |
| 10 | lack void | 缺少void |
| 11 | lack while | 缺少while |
| 12 | lack case | 缺少case |
| 13 | lack int OR char | 缺少int或char |
| 15 | lack return | 缺少return |
| 17 | lack scanf | 缺少scanf |
| 18 | lack printf | 缺少printf |
| 19 | Illegal unsigned integer | 非法无符号整数 |
| 20 | Illegal identifier | 非法标识符 |
| 21 | Illegal assignment | 非法赋值 |
| 22 | Constant error | 常量错误 |
| 23 | String does not meet the requirements | 字符串不符要求 |
| 27 | Illegal return value | 非法返回值 |
| 28 | Array access cross-border | 数组越界 |
| 30 | Wrong Func Call | 错误函数调用 |

错误处理的方式就是跳过错误代码段，具体函数是Jump函数。

|  |  |
| --- | --- |
| 处理序号 | 错误处理 |
| 1 | 跳到下一个分号 |
| 2 | 跳到下一个逗号或分号 |
| 3 | 跳到下一个int或char或void |
| 4 | 跳到下一个} |
| 5 | 跳到下一个= |
| 6 | 跳到下一个const或int或char或void |
| 7 | 跳到下一个const或int或char或<语句>头符号 |
| 8 | 跳到下一个分号的后一个位置 |

## 三．操作说明

### 1．运行环境

VS2010 Mars4.5

### 2．操作步骤

|  |
| --- |
| 操作步骤 |
| 1.打开VS2010和Mars4.5 |
| 2.编译运行编译器代码 |
| 3.输入测试文件路径 |
| 4.按下回车，开始编译并生成目标代码 |
| 5.打开目标代码文件并复制 |
| 6.粘贴到Mars4.5 |
| 7.运行mips汇编代码 |
| 8.看结果框中的结果是否正确 |

## 四．测试报告

### 1．测试程序及测试结果

|  |
| --- |
| 测试程序1 |
| Const char ch1='e', ch2='n', ch3='d';  const int num1 = 1,num2 = 2,\_num = 10,negnum=-1024;  int fbnum,fbResult;  char m [5];  voId nullfunc()  { }  void count()  {  char c;  char name;  int letters,star;  int digits;  int others;  letters=0;  star=0;  digits=0;  others=0;  priNtf("scanf some words:(end is /)");  do  {  scanf(C);  if(c>='a')  {  if(c<='z')  letters=letters+1;  }  if(c>='A')  {  if(c<='Z')  letters=letters+1;  }  if(c>='0')  {  if(c<='9')  digits=digits+1;  }  if(c=='\*')  star=star+1;  }while(c!='/')  printf("the num of letters is",letters);  printf("the num of digits is",digits);  printf("the num of star is",star);  }  int age(int n)  {  const int ten = 10;  int c;  if(n)  {  if(n==1)  c=(ten);  if(n!=1)  c=age(n-1)+2;  }  return (c);  }  int fact(int j)  {  int sum;  if (j == 0)  sum = 1;  if (j!=0)  sum = j\*fact(j - 1);  return (sum);  }  void fact5()  {  const int worong = -2;  int i;  i = 0;  do  {  printf("answers are: ",i);  printf(fact(i));  i = i + 1;  } while (i<6)  }  int fibonaci(int i)  {  if(i == 0)  {  return (0);  }  if(i == 1)  {  return (1);  }  return (fibonaci(i-1) + fibonaci(i-2));  }  void main()  {  const int worong = -2,arr=8;  char a[10],\_[9];  int i,m;  sCanf(i);  switch(i)  {  case -1:  {  m=age(-3+8);  printf(m);  }  case 3:fact5();  case 4:  {  printf("scanf fbnum!");  scanf(fbnum);  fbResult = fibonaci(fbnum);  printf(fbResult);  }  case 5:count();  }  printf("---The End---");  return;  } |

|  |
| --- |
| 测试程序1——结果 |
| 》》如果输入-1，结果为  18  ---The End---  》》如果输入3，结果为  3  answers are:  0  1  answers are:  1  1  answers are:  2  2  answers are:  3  6  answers are:  4  24  answers are:  5  120  ---The End---  》》如果输入4，结果为  4  scanf fbnum!  10  55  ---The End---  》》如果输入5，结果为  5  scanf some words:(end is /)  4  5  t  W  q  3  \*  6  t  A  /the num of letters is  5  the num of digits is  4  the num of star is  1  ---The End--- |

|  |
| --- |
| 测试程序2（测试嵌套函数调用） |
| int add**(**int a**,**int b**)**  **{**  **return(**a**+**b**);**  **}**  void main**()**  **{**  int k**;**  k **=** add**(**0**,**add**(**1**+**2**,**3**+**4**))+**add**(**1**+**1**,**add**(**1**,**1**)+**1**);**  printf**(**k**);**  **}** |

|  |
| --- |
| 测试程序2——结果 |
| 15 |

|  |
| --- |
| 测试程序3（函数调用及四则运算） |
| const int m **=** 9**,**nu **=** 6**;**  int p**;**  int funTest**(**int i**)**  **{**  **if(**i**>**1**)**  printf**(**"i is bigger than 1"**);**  p**=**1**;**  p**=**p**+**m**;**  printf**(**p**);**  printf**(**'a'**);**  printf**(**"test printf:"**);**  printf**(+-**2**);**  printf**(+**2**+-**2**);**  printf**(**0**);**  printf**(-**5**);**  **return(**3**);**  **}**  void main**()**  **{**  const int b **=**3**;**  int c**,**d**,**e**;**  c**=**1**;**  d**=**2**;**  e**=**2**;**  p**=**funtest**(**8**);**  printf**(**p**);**  p**=**b**+**c**-**d**\***e**;**  printf**(**p**);**  **}** |

|  |
| --- |
| 测试程序3——结果 |
| i is bigger than 1  10  a  test printf:  -2  0  0  -5  3  0  -- program is finished running -- |

|  |
| --- |
| 测试程序4 |
| const int mm**=-**2**;**  const int nn **=** **-**9**,**k**=-**8**;**  void main**()**  **{**  int e**,**key**;**  scanf**(**e**);**  **switch(**e**)**  **{**  **case** **-**4**:**  **{**  scanf**(**key**);**  **switch(**key**\***1**)**  **{**  **case** 0**:**printf**(**"key is 0"**);**  **case** 2**:**printf**(**"key is 2"**);**  **}**  **}**  **case** 5**:**printf**(**"this is 1"**);**  **case** 6**:**printf**(**e**);**  **case** **-**8**:**printf**(**nn**);**  **}**  **}** |

|  |
| --- |
| 测试程序4——结果 |
| \*\*\*\* user input : -4  \*\*\*\* user input : 2  key is 2  -- program is finished running -- |

|  |
| --- |
| 测试程序5（int/char类型的输出，局部变量与全局重名） |
| const int m **=** 9**,**nu **=** 6**;**  char k**;**  void main**()**  **{**  char m**;**  printf**(**'a'**-**'+'**+** '+'**);**  printf**(**'a'**+**1**);**  printf**(**'a'**-**'+'**+** 1**);**  m**=**'a'**+**1**;**  printf**(**m**);**  k**=**'a'**+**1**;**  printf**(**k**);**  **}** |

|  |
| --- |
| 测试程序5——结果 |
| a  98  55  b  b  -- program is finished running -- |

|  |
| --- |
| 测试程序6（对数组循环赋值） |
| char a**[**10**];**  void main**()**  **{**  int i**;**  int b**[**8**];**  char a**[**12**];**  i **=**0 **;**  **do**  **{**  a**[**i**]=**'a'**+**i**;**  i**=**i**+**1**;**  **}while(**i**<**5**)**  b**[**3**]** **=** 9**;**  i **=** b**[**3**];**  printf**(**i**);**  printf**(**a**[**0**]);**  printf**(**a**[**1**]);**  printf**(**a**[**2**]);**  printf**(**a**[**3**]);**  printf**(**a**[**4**]);**  **}** |

|  |
| --- |
| 测试程序6——结果 |
| 9  a  b  c  d  e  -- program is finished running -- |

|  |
| --- |
| 错误测试程序7（数组越界，调用未定义函数，使用未定义变量） |
| const int a**=**3**;**  char arr**;**  int sz**[**180**];**  void test**()**  **{**  int arr**[**100**];**  int sz**;**  int a**[**10**];**  a**[**11**]=**9**;**  a**[**10**]=**8**;**  a**[**100**-**98**]=**2**;**  **}**  void main**()**  **{**  int w**;**  funa**();**  w **=** funb**();**  w **=** arr**[**2**];**  w **=** sz**;**  **}** |

|  |
| --- |
| 错误测试程序7——结果 |
| Error28 Array access cross-border  错误可能在第 9 行 类型是Num 名字是11  Error28 Array access cross-border  错误可能在第 10 行 类型是Num 名字是10  Error6 lack ; Semi  错误可能在第 15 行 类型是Semi 名字是;  跳到下一个分号  -\*-\*-\*-\*-\*-\*-\*-\*-\*-  有返回值的函数调用中函数未定义：第17行funb  Error30 Wrong Func Call  错误可能在第 17 行 类型是IdSym 名字是funb  数组未定义：第18行arr  标识符未定义：第19行sz |

|  |
| --- |
| 错误测试程序8（缺类型、分号、未定义标识符） |
| const h**=**'a'**;**  int m**,**n  int fun**()**  **{**  int a**;**  a **=** 1**;**  a**=**a**+**5**;**  **return** **(**c**);**  **}**  VOID main**()**  **{**  fun**();**  **}** |

|  |
| --- |
| 错误测试程序8——结果 |
| Error13 lack int OR char  错误可能在第 1 行 类型是constSym 名字是const  跳到下一个分号  -\*-\*-\*-\*-\*-\*-\*-\*-\*-  Error6 lack ; Semi  错误可能在第 2 行 类型是IdSym 名字是n  跳到下一个int或char或void  -\*-\*-\*-\*-\*-\*-\*-\*-\*-  标识符未定义：第8行c  Error27 Illegal return value  错误可能在第 8 行 类型是IdSym 名字是c |

|  |
| --- |
| 错误测试程序9（重复定义） |
| const int a**=**1**;**  const int a**=**2**,**a**=**5**;**  const char a **=** 'a'**;**  int a**;**  char a**;**  void fun**()**  **{**  int a**;**  char a**,**fun**;**  **}**  void main**()**  **{**  const char fun **=** '6'**;**  const int a **=** 7**;**  int a**,**b**;**  char a**;**  char b**;**  **}** |

|  |
| --- |
| 错误测试程序9——结果 |
| 第2行与a有重名  第2行与a有重名  第3行与a有重名  第4行与a有重名  第5行与a有重名  第9行与a有重名  第9行与函数名fun重名  第13行与函数名fun重名  第15行与a有重名  第16行与a有重名  第17行与b有重名 |

|  |
| --- |
| 错误测试程序10（int与char类型的转换） |
| int m**;**  char fun**()**  **{**  **return(**'a'**);**  **}**  void main**()**  **{**  char arr**[**20**];**  char n**;**  n **=** 'w'**+**'a'**;**  arr**[**3**]=**'q'**;**  arr**[**4**]=**arr**[**3**]+**1**;**  arr**[**4**]=**'a'**+**'\*'**;**  m **=** arr**[**3**];**  m **=** fun**();**  **}** |

|  |
| --- |
| 错误测试程序10——结果 |
| 第10行'a' 此处有int向char类型的转换  第12行1 此处有int向char类型的转换  第13行'\*' 此处有int向char类型的转换 |

|  |
| --- |
| 错误测试程序11（函数有无返回语句） |
| int fun**()**  **{**  char a**;**  a **=** 3**;**  **}**  void fun2**()**  **{**  char b**;**  int c**;**  b **=** 'b'**;**  **return** **(**b**);**  **}**  void main**()**  **{**  int d**,**e**;**  d **=** fun**();**  e **=** fun2**();**  printf**(**d**);**  printf**(**e**);**  **}** |

|  |
| --- |
| 错误测试程序11——结果 |
| 第4行3 此处有int向char类型的转换  Error15 lack return  错误可能在第 4 行 类型是Semi 名字是;  Error27 Illegal return value  错误可能在第 11 行 类型是Semi 名字是;  Error30 Wrong Func Call  错误可能在第 17 行 类型是IdSym 名字是fun2 |

### 2．测试结果分析

|  |  |
| --- | --- |
| 测试程序 | 覆盖语法成分 |
| 1 | 常量说明，变量说明，赋值语句，写语句，do-while循环，if语句的嵌套，函数自身递归调用，函数对其他函数的调用，switch语句 |
| 2 | 嵌套函数调用，赋返回值，写语句 |
| 3 | if语句，写语句，main内调用函数，四则运算 |
| 4 | 常量说明，函数内的变量说明，读语句，switch语句嵌套使用，写语句 |
| 5 | 常量说明，变量说明，写语句，赋值语句 |
| 6 | 变量说明，赋值语句，写语句，do-while循环 |
| 7 | 常量说明，变量说明，无返回值的函数定义，赋值语句，函数的调用 |
| 8 | 常量说明，变量说明，有返回值的函数定义，赋值语句，函数调用 |
| 9 | 常量说明，变量说明，无返回值函数定义 |
| 10 | 变量说明，有返回值函数定义，赋值语句，函数调用 |
| 11 | 有返回值函数定义，无返回值函数定义，返回语句，函数调用，写语句 |

## 五．总结感想

学习编译课程设计以来，我的受益良多。

一、面对bug的心态越来越平和，处理bug的耐心程度有了大幅提高。我以前看到bug就会非常烦躁，完全不想去改。现在当我看到一个bug被亲手改好，内心的喜悦是无与伦比的。

二、凡事不要拖延。课程安排了每周一个DDL，我们的进度应当快于DDL，这样才不会出现交作业前一天代码还没改好或是还未写好的情况。当一件事情是我们必须要去做的，何不早早做好，尽善尽美，而不是拖延到最后关头才去实现。事情是不会摆在那里自己做好，而是需要我们身体力行的去做。

三、多多测试。很多bug都是在大佬们的超级厉害的测试程序中被发现的。集思广益总是会帮我们发现我们想不到的问题。

四、多方求助。关于文法的理解总会不太深刻，我们可以利用好judge平台，多多提问。但是求助不等于抄袭。

五、经常关注信息。比如judge上发布的信息，PPT提到的信息。目标代码运行考核成功、参加测试程序征集讨论就可以获得一次额外测试机会，这些都是非常有用的信息。Judge上发布的关于输出的要求、重名的定义等都非常重要，倘若没有去留意，就会在测试中白白丢分。