

**编译原理实验课程报告**

**题目：MiniC语言编译器设计与实现**

专业班级：

学 号：

姓 名：

指导教师：

报告日期： 2023年 月 日

网络空间安全学院

**要 求**

1、实验代码及报告为本人独立完成，内容真实。如发现抄袭，成绩无效；如果引用资料，需将资料列入报告末尾的参考文献，参考文献格式按华中科技大学本科毕业论文规范，并在正文中标注参考文献序号；

2、按编译原理实验任务，内容应包含：工具入门、词法分析、语法分析、语义分析及中间代码生成、目标代码生成；

3、报告中简单说明遇到的问题及解决问题的思路，特别是与众不同的、独特的部分；对设计实现中遇到的问题、解决进行记录；根据实验内容，结合能力训练的目标，对实验进行总结；完成自我评价；

4、评分标准：5个主要实验环节按任务要求完成；采用的方法合适、设计合理；能体出研究能力、工具选择、工具开发、自主学习相关的能力；报告条理清晰、语句通顺、格式规范。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4.2研究能力 | 5.2 工具选择 | 5.3工具开发 | 12.2 自主学习 | 总分 |
| 25 | 40 | 10 | 25 | 100 |
|  |  |  |  |  |

**目 录**

[一、实验过程记录 1](#_Toc1762285680)

[（一） Flex & Bison入门 1](#_Toc1784010665)

[（二） MiniC 语法（Flex） 1](#_Toc346359532)

[（三） MiniC 语法分析及语法树生成 2](#_Toc397402409)

[（四） MiniC 语义分析及中间代码生成 4](#_Toc243549572)

[（五） MiniC 代码优化及目标代码生成 6](#_Toc1916205565)

[二、实验心得 7](#_Toc1100770095)

[三、实验目标达成度的自我评价 8](#_Toc218147824)

[四、实验建议 9](#_Toc1633645797)

[参考文献 10](#_Toc1073021978)

## 

## 一、实验过程记录

1. **Flex & Bison入门**
   * 1. Flex规则匹配顺序

Flex在进行匹配时，若存在多条可用的匹配规则，只会选中其中一条规则进行匹配，例如实验中的示例规则：

1. a\*b             {printf("1");}
2. ca              {printf("2");}
3. a\*ca\*           {printf("3");}

输入的符号串为abcaacacaaabbaaabcaaca，运行flex匹配后输出结果为132311132，根据输出结果可以得到匹配的情况为ab / caa / ca / caaa / b / b / aaab / caa / ca。

观察到caa串的匹配结果为3，而不是2，根据该结果猜测flex并非遇到匹配结果及输出，似乎会寻找最长的匹配。caa串后紧接着的ca串的匹配结果是2，而不是3，猜测除了上述最长匹配规则外，还有按照规则顺序进行匹配的规则。

Flex文档中对匹配规则的描述文档如下：

If it finds more than one match, it takes the one matching the most text (for trailing context rules, this includes the length of the trailing part, even though it will then be returned to the input). If it finds two or more matches of the same length, the rule listed first in the flex input file is chosen.[3]

文档中说明了flex进行匹配时会寻找最长的匹配，当有多个长度相同的匹配时，选择第一个出现的匹配，根据该规则对上述输入符号串进行匹配，匹配情况为ab / caa / ca / caaa / b / b / aaab / caa / ca，得到的结果与flex运行结果相同。

1. **MiniC 语法（Flex）**
   * 1. Flx minic词法分析（二）

实验要求能够识别浮点数，八进制数和十六进制数，并能够识别非法八进制数与十六进制数，例如08、0xGF2。

对于十进制整数，如果值非0，则第一个出现的数字不能为0，只能为1-9，根据该规则，构造识别十进制整数的正则表达式如下：

1. **INT**            [\-]?[1-9][0-9]\*|0

八进制整数由0开头，0接下来的一个数字不能为0，因此只能为1-7，后续接多个0-7，根据该规则，构造识别八进制整数的正则表达式如下：

1. **OCTINT**        [\-]?0[1-7][0-7]\*

十六进制数的格式与八进制数相似，由0x开头，接下来的第一个数字为1-9或a-f或A-F，后续接多个0-9或a-f或A-F，根据该规则，构造识别十六进制整数的正则表达式如下：

1. **HEXINT**     [\-]?0x[1-9a-fA-F][0-9a-fA-F]\*

浮点数共有两种格式，一种为常见的点分十进制格式，形如1.2，第二种为科学计数法表示，形如1.05e5。根据两种格式构造出识别浮点数的正则表达式如下：

1. **FLOAT1**      [\-]?[0-9]+\.[0-9]+
2. **FLOAT2**      [\-]?[0-9]+\.[0-9]+e[\-]?[1-9][0-9]\*

minic中标识符的定义为由大小写字母开头，后接多个大小写字母或数字，根据标识符的定义，构造识别标识符的正则表达式如下：

1. **IDENT**            [A-Za-z][A-Za-z0-9]\*

编写flex识别数字、标识符的匹配规则：

1. {IDENT}       {flexout("ID", yytext);}
2. {INT}         {flexout("INT", yytext);}
3. {OCTINT}      {flexout("INT", yytext);}
4. {HEXINT}      {flexout("INT", yytext);}
5. {FLOAT1}      {flexout("FLOAT", yytext);}
6. {FLOAT2}      {flexout("FLOAT", yytext);}
7. **MiniC 语法分析及语法树生成**
   * 1. Bison工作原理及移进规约冲突解决

实验给出的bison规则代码如下：

1. %**union**
2. {
3. **int** ival;
4. **char** \*sval;
5. }
6. %token <ival>NUM
7. %token <sval>STR
8. %nterm <ival>exp
9. %nterm <sval>useless
10. %left '+'
11. %%
12. exp:
13. exp '+' exp
14. |exp '-' exp
15. |NUM
16. ;
17. useless: STR;
18. %%

使用bison处理规则后得到如下的错误提示：

1. Nonterminals useless in grammar
2. useless
3. Terminals unused in grammar
4. STR
5. Rules useless in grammar
6. 4 useless: STR
7. State 6 conflicts: 1 shift/reduce
8. State 7 conflicts: 2 shift/reduce

第1、2行提示useless非终结符未使用，因此需要移除规则中的第9行对useless非终结符的定义；第3、4行提示STR终结符未使用，因此需要移除规则中的第7行对STR终结符的定义；第5、6行提示规则useless: STR未使用，因此需要移除规则中的第17行多于规则。

输出中第7行提示状态6存在移入/规约冲突，第8行提示状态7存在移入/规约冲突，查看状态6与状态7：

1. State 6
2. 1 exp: exp . '+' exp
3. 1    | exp '+' exp .
4. 2    | exp . '-' exp
5. '-'  shift, and go to state 5
6. '-'       [reduce **using** rule 1 (exp)]
7. $**default**  reduce **using** rule 1 (exp)
8. Conflict between rule 1 and token '+' resolved as reduce (%left '+').
10. State 7
11. 1 exp: exp . '+' exp
12. 2    | exp . '-' exp
13. 2    | exp '-' exp .
14. '+'  shift, and go to state 4
15. '-'  shift, and go to state 5
16. '+'       [reduce **using** rule 2 (exp)]
17. '-'       [reduce **using** rule 2 (exp)]
18. $**default**  reduce **using** rule 2 (exp)

根据状态6和7的内容可知，当识别到’+’或’-’时，移入和规约两种操作都可用，存在冲突，bison使用移入的方式解决该冲突，根据bison文档[4]，可以为识别符号添加结合性解决该冲突情况，在规则的第10行后添加%left ‘-’，规定’-’为左结合性即可解决冲突。

* + 1. Bison 语法规则构造（二）冲突状态解决

语法规则构造参考Mini-C语法规则[5]，完成语法规则的构造后，使用bison处理parser.y发现存在大量的移入/规约冲突，分析parser.y文件，观察到未定义终结符的结合性，根据前面实验中对移入/规约冲突的解决方式，根据Mini-C语法规则，对终结符的结合性做如下约定：

1. %left COMMA
2. %left ASSIGNOP PLUSASS MINUSASS STARASS DIVASS MODASS
3. %left OR
4. %left AND
5. %left RELOP
6. %left PLUS MINUS
7. %left STAR DIV MOD
8. %right UMINUS NOT UPLUSPLUS UMINUSMINUS
9. %left PLUSPLUS MINUSMINUS
10. %right DOT
11. %left LB
13. %nonassoc LOWER\_THEN\_ELSE
14. %nonassoc ELSE

修改终结符的结合性后，重新使用bison处理parser.y，解决了规则中存在的移入/规约冲突，编译得到的扫描器能够准确识别出minic代码中的语法符号和语法规则。

* + 1. Bison输出语法树

完成此法分析和语法分析后，扫描代码可生成代码的语法树，语法树以Program为根节点。语法树中所有的节点所属的类都继承自Node类，打印输出语法树时，调用节点的parser方法输出节点信息和递归地输出所有子节点信息，根据定义的语法规则，补全所有节点类的parser方法，格式化输出节点的语法树。

在补全各节点类的parser方法时，需要注意部分类有多种规约规则，其子节点不一定存在，或子节点的数量可能不同，需要根据具体情况判断是否输出子节点语法树，例如函数调用节点NMethodCall::parse()，需要根据函数调用参数是否存在以及参数的个数决定如何递归地输出参数节点的语法树。

使用yyparser处理输入minic代码后，生成代码的完整语法树，调用根节点的parser方法，即可输出完整的语法树。

1. **MiniC 语义分析及中间代码生成**
   * 1. LLVM IR初识

LLVM IR包含常量、变量、表达式计算、函数、分支指令等多个内容。

常量包括基础的布尔常量、整形常量等简单常量，以及结构常量、数组常量等复杂常量，常量与具体的类型有关，在使用时需要指明其类型，例如i32 0。

变量包括全局变量和局部变量，全局变量使用@开头，局部变量使用%开头，命名的变量使用标识符表示，例如%foo、@DivisionByZero，未命名的变量使用数字表示，例如%12、@2。每一个变量都需要使用alloca申请内存空间，例如，为变量%1申请空间：

1. %1 = alloca i32, align 4

变量的使用需要使用store/load操作，store将值写到变量中，load读取变量的值，例如，将0写入变量%1，再将变量%1的值加载到临时变量%2中：

1. store i32 0, i32\* %1, align 4
2. %2 = load i32, i32\* %1, align 4

表达式计算主要为二元计算，例如乘法mul和加法add：

1. %resultMul = mul i32 %x, 8
2. %0 = add i32 %x, %x
3. %1 = add i32 %0, %0
4. %resultAdd = add i32 %1, %1

函数声明格式和函数定义格式如下：

1. declare dso\_local i32 @putchar(i32) #1
3. define dso\_local i32 @inc() #0 {
5. }

其中的dso\_local和#0 #1为函数的描述和修饰字段。函数声明和定义的方式与高级语言接近，需要指明函数的返回类型，如i32，函数名命名规则与全局变量类似以@开始。

函数调用的格式为call 返回类型 @id(参数)，例如putchar的调用对应的代码：

1. %13 = call i32 @putchar(i32 %12)

LLVM IR的分支跳转主要通过icmp指令和br指令实现，icmp指令先进行比较，根据比较结果利用br指令实现跳转。icmp指令后接标志位，如sgt、eq等，br指令分为条件转移和无条件转移两种，例如如下代码，第1行使用icmp eq比较变量%4是否等于97，第2行根据比较结果进行条件跳转，当比较结果为真时，跳转到标签6，比较结果为假则跳转到标签10，第6行和第10行为无条件跳转，直接跳转到标签10。

1. %5 = icmp eq i32 %4, 97
2. br i1 %5, label %6, label %8
4. 6:
5. %7 = call i32 @putchar(i32 89)
6. br label %10
8. 8:
9. %9 = call i32 @putchar(i32 78)
10. br label %10
12. 10:
13. %11 = call i32 @putchar(i32 10)
14. store i32 0, i32\* %2, align 4
15. ret i32 0
    * 1. LLVM IR API

实验中通过手动调用LLVM API，逐条生产IR代码翻译提供的样例C语言代码。

使用ConstantInt::get创建按常量， builder->CreateAlloca创建变量，变量的创建需要传入变量的类型，例如使用Type::getInt32Ty(\*theContext)指定变量类型为i32，表达式的计算使用CreateAdd、CreateSub、CreateMul等API，表达式计算之前需要使用CreateLoad加载变量，计算后使用CreateStore储存变量。函数的创建使用Function::Create，函数创建需要指定返回类型和参数的类型，函数的调用使用builder->CreateRet，函数调用需要传入调用的函数和函数的参数列表。分支跳转包括条件跳转builder->CreateCondBr，无条件跳转builder->CreateBr，条件跳转需要指定跳转的条件，为真时跳转的块以及为假时跳转的块，无条件跳转需要指定跳转的块。

通过选择合适的API，生成与实例C语言代码对应的IR中间代码，在API的使用时，关于API的用途，参数情况参考LLVM官方文档[6]。

运行生成的IR中间代码进行测试，其运行正常，并且能够正确实现原C语言代码的功能。

* + 1. 语义分析与中间代码生成

该实验环节与语法数生成环节类似，通过补全节点类的codegen方法，在遍历语法树时，生成该节点对应的IR代码，并递归遍历子节点，调用子节点codegen生成子节点的IR代码。

编写节点的codegen方法时，需要根据节点的定义，选择合适的LLVM API生成对应语法结果的中间代码，还需要根据语法树结构创建代码块，设计分支结构，定义函数等，需要根据节点的具体情况判断是否递归遍历子节点。

使用yyparser处理输入minic代码后，生成代码的完整语法树，调用根节点的codegen方法，即可遍历整个语法树，生成minic代码对应的IR代码，使用llvm即可运行生成的IR中间代码。

1. **MiniC 代码优化及目标代码生成**
   * 1. LLVM PASS

该实验内容是对前面实验编写的MiniC编译器增加LLVM优化功能。

在语义分析任务中的astnode.cpp文件中，调用函数InitializeModuleAndPassManager()启用PASS优化，未开启PASS优化时，编译生成的IR代码为82行，经过优化后编译生成的IR代码长度为50行，开启PASS优化后内存操作指令的数量有了明显的减少。

除了编译生成IR阶段的PASS优化，还可以使用opt -S -mem2reg对中间代码进行mem2reg优化，将常用变量优化为寄存器变量，减少程序运行时访问内存的次数，提高程序运行效率。

对中间IR代码进行优化后，使用llc将中间代码编译为目标平台的汇编代码，最后使用clang将汇编代码编译连接得到minic代码对应的可执行文件。

## 二、实验心得

在本次编译原理实验中，通过完成词法分析、语法分析、语义分析、中间代码生成和目标代码生成优化这几个过程，逐步构建了MiniC语言的编译器，能够编译MiniC语言生成LLVM中间代码，并使用LLVM提供的API对中间代码的生成过程进行优化，最后使用llc、clang等工具生成可在目标平台运行的二进制文件。

实验选用的MiniC语言是C语言的一个子集，相比完整的C语言，语法和特性更为简洁明确，减少了学习和实验的复杂度，能够更专注于核心的编译原理算法和实现细节，更快地获取实验结果。

词法分析和语法分析的步骤中使用了Flex和Bison作为词法分析器和语法分析器生成工具，flex和bison提供了强大的词法分析和语法分析功能，大大简化了编译器开发的过程，避免在词法分析器和语法分析器的开发上耗费过多的时间，使实验的精力能够集中在编译原理的算法上，而非底层工具的实现细节上。由于在本次实验前未使用过flex和bison工具，对于其使用方法和语法规则了解较少，在实验过程中除了参考实验文档外，还需要反复查阅官方文档，了解编写词法分析、语法分析代码的细节，通过官方文档、互联网资料、AI工具等，学习了flex和bison的基础语法、了解这两种工具的使用方法，能够通过编写词法分析和语法分析代码处理指定的MiniC代码，生成对应的语法树。

中间代码生成过程中，使用LLVM提供的C++ API，将语法树翻译为LLVM中间语言。实验中需要大量使用C++的面向对象编程的内容，并且根据语法树中节点的类型，使用LLVM API编写合适的中间代码生成规则，对于如何选择需要的API，以及API的详细使用方式，也需要查询相应的官方文档，搜索和归纳互联网资料。

本次实验除了理论课中所学的编译原理知识外，还涉及多种工具的使用，通过设计简单的编译器，将抽象的理论知识变为更直观易懂的代码，加深对编译原理理论知识的认识，学习了如何将理论知识融入到实际使用中，在学习flex、bison、llvm工具使用时，需要大量查阅相关资料，归纳整理网络信息提取出自己需要的内容，加强了自己的信息搜集能力。

## 三、实验目标达成度的自我评价

通过实验，结合前面实验心得中的内容，在下面的表格中，完成自我评价。

|  |  |  |
| --- | --- | --- |
| 毕业目标 | 自我评价的具体内容 | 目标达成的满意度 自评 ☑标记 |
| 4.2能够基于科学原理和方法，根据需求选择路线，设计方案； | 在实验过程中，包括词法、语法、语义分析、中间代码生成、目标代码优化过程中，需要根据课程中学习的编译理论课的原理和方法，确定合理的完成实验的路线，设计方案。包括：语言语法结构的取舍；选择分析路径，自下而上分析/自上而下分析；为了完成最终的代码生成，是否拟定/跟随完成了前序的学习任务（工具及相应语言的学习）等。 | □非常满意  ☑满意  □一般  □不满意 |
| 5.2选择、使用现代工具设计、预测、模拟与实现，分析局限； | 实验中，要求使用现代工具，如新的词法工具Flex、语法工具Bison及中间代码框架LLVM的内容，完成实验的设计、实现；对相应工具实现时的局限性，进行适当的分析；甚至拟定出进一步完善的方案或方向。 | ☑非常满意  □满意  □一般  □不满意 |
| 5.3开发满足特定需求的现代工具，分析其局限 | 编译原理实验中，引入了一个C语言的子集，并进行了相对快速的编译实验，本质上，为今后开发领域语言或者代码优化、代码分析工作，进行了准备。能对实现的简单编译器原型，进行局限性分析。 | □非常满意  ☑满意  □一般  □不满意 |
| 12.2获取和职业发展需要的自主学习的能力，并表现出相应的成效。 | 通过阅读实验任务给的资料（英文Flex、Bison、LLVM网站）及自行搜索、整理、归纳互联网上的资源，见参考文献列表及相关文献在正文中的合适引用；通过完成铺垫关卡任务，快速掌握三种语言的基本功能；对于给出的工程问题（编译器构造）中，利用学习收获的知识，设计出相应的方案并实现其原型，从而获取了和职业发展需要的自主学习能力。 | □非常满意  ☑满意  □一般  □不满意 |

## 实验建议

实验通过数十个关卡，从词法分析开始逐步完成MiniC编译器的设计，很好地与理论课知识相对应，并且实验使用了多种成熟的工具，减小了编译器涉及的难度，但实验中也存在部分问题。首先是实验的任务量较大，flex、bison、llvm以及c++面向对象的部分都是未学习的内容，需要耗费大量的时间了解这些工具的使用，虽然实验中提供了实验文档，但文档内容过于碎片化，依然需要查询官方文档或网络上的其他资料，建议实验采用更轻量级的工具链。其次实验基于educoder平台，平台能够对编写后的代码进行评测，但是出现错误情况时，平台的反馈不足，并且平台难以调试，本地部署开发环境时，存在工具链版本差异，导致本地结果与平台结果存在差异。

## 参考文献

1. 编译原理（第4版） 电子工业出版社
2. 编译原理实践与指导教程 机械工业出版社
3. [Flex, version 2.5 How the input is matched](https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_mono/flex.html" \l "TOC8)
4. [Bison 3.8.1 Understanding Your Parser](https://www.gnu.org/software/bison/manual/bison.html" \l "Understanding)
5. [Mini-C语法规则](https://data.educoder.net/api/attachments/2204585?disposition=inline)
6. [LLVM Language Reference Manual](https://llvm.org/docs/LangRef.html)