華中科技大學

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：网安 2002 班

学　　号：U202012043

姓　　名：范启航

指导教师：刘铭

报告日期：2022 年 12 月 10 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；每个任务进行归纳、小结；最后，对整个实验进行总结，列出主要优点、不足；

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；能体出分析问题、灵活运用知识解决实际问题的能力；报告条理清晰、语句通顺、格式规范；

工具入门	词法分析	语法分析	语义分析及中间代码生成	优化及目标代码生成	格式规范	总分
10	20	30	20	10	10	100

目录

一、实验过程

1.1 Flex & Bison 工具入门

（1）实验内容

Lex 是 Lexical Compiler 的缩写，是 Unix 环境下非常著名的工具，主要功能是生成一个词法分析器(scanner)的 C 源码，描述规则采用正则表达式(regular expression)。本次实验使用 Lex 对 minic 进行词分析。

（2）实验过程 task101: 对输入文件进行行数，字符树分析。

实现方法：规则段：匹配字符/n 则行数+1, 其他字符则，字符数+1，规则段如下图所示：

```
\n ++num_lines;
.  ++num_chars;
```

图 1-1 匹配规则

分析结束时输出字符数与行数，程序输出部分如下图所示：

```
yylex();
printf("Lines=%d,Chars=%d\n",num_lines,num_chars);
```

图 1-2 程序分析输出

task102：实现简单的 toy 语言的词法分析，实现对整数，小数，部分关键字，标识符，操作符号的识别，输出不能识别的符号。匹配规则如下图所示：

```
DIGIT [0-9]+
ID [a-z][a-z0-9]*
%%
{DIGIT}             {printf( "An integer: %s (%d)\n", yytext,atoi( yytext ) );}
{DIGIT}+"."{DIGIT}?      {printf( "A float: %s (%g)\n", yytext,atof( yytext ) );
}
if|then|begin|end|procedure|function {printf( "A keyword: %s\n", yytext );}
{ID} printf( "An identifier: %s\n", yytext );
"+"|"-"|"*"|"/"         printf( "An operator: %s\n", yytext );
"{"[^}\n]*"}"   /* eat up one-line comments */
[ \t\n]+                /* eat up whitespace */
.       printf( "Unrecognized character: %s\n", yytext );
```

图 1-3 匹配规则

测试结果如下图所示：

```
cse@s-614:~/miniC/lab1/task102$ ./scanner case0.in
An identifier: abc
An integer: 123 (123)
A float: 12.3 (12.3)
An identifier: abc123
An integer: 123 (123)
An identifier: abc
A keyword: if
An identifier: else
A keyword: then
An integer: 12 (12)
An integer: 3 (3)
```

图 1-4 测试结果

task103：了解 flex 规则在匹配时，如果多条规则都匹配成功， yylex 会选择匹配长度最长的那条规则，如果有匹配长度相等的规则，则选择排在最前面的规则。

预测结果：132311132 测试结果如下图所示：

```
cse@s-614:~/miniC/lab1/task103$ ./scanner case0.in
132311132cse@s-614:~/miniC/lab1/task103$
```

图 1-5 测试结果

task104：利用 flex 工具生成 PL 语言的词法分析器

实现方法：编写对应的匹配规则，匹配规则部分如下图所示：

```
INTCON          [\-]?[1-9][0-9]*|0
IDENT           [A-Za-z][A-Za-z0-9]*
CHARCON         [^\']*


%%
"of"            {printf("%s: OFSYM\n", yytext);}
"array"         {printf("%s: ARRAYSYM\n", yytext);}
"program"       {printf("%s: PROGRAMSYM\n", yytext);}
"mod"           {printf("%s: MODSYM\n", yytext);}
"and"           {printf("%s: ANDSYM\n", yytext);}
"or"            {printf("%s: ORSYM\n", yytext);}
"not"           {printf("%s: NOTSYM\n", yytext);}
"begin"         {printf("%s: BEGINSYM\n", yytext);}
"end"           {printf("%s: ENDSYM\n", yytext);}
"if"            {printf("%s: IFSYM\n", yytext);}
"then"          {printf("%s: THENSYM\n", yytext);}
"else"          {printf("%s: ELSESYM\n", yytext);}
"while"         {printf("%s: WHILESYM\n", yytext);}
"do"            {printf("%s: DOSYM\n", yytext);}
"call"          {printf("%s: CALLSYM\n", yytext);}
"const"         {printf("%s: CONSTSYM\n", yytext);}
"type"          {printf("%s: TYPESYM\n", yytext);}
"var"           {printf("%s: VARSYM\n", yytext);}
"procedure"     {printf("%s: PROCSYM\n", yytext);}
```

图 1-6 匹配规则（部分）

task105：利用 YACC/Bison 构建一个逆波兰符号计算器。

实现方法：根据逆波兰式计算规则编写 bison BNF 范式。语法规则如下图所示：

```
line:
  '\n'
| exp '\n'      { printf ("%.10g\n", $1); }
;

exp:
  NUM           { $$ = $1;           }
 /* begin */
| exp exp '+'   { $$=$1+$2;}
| exp exp '-'   { $$=$1-$2;}
| exp exp '^'   { $$=pow($1, $2);}
| exp exp '/'   { $$=$1/$2;}
| exp exp '*'   { $$=$1*$2;}
| exp 'n'       { $$=-$1;}
 /* end */
;
```

图 1-7 语法规则

task106：继续使用 Bison，完成中缀式计算器的语法规则设计。

语法规则如下图所示：

```
 /* begin */
%token LP RP

 /* end */
%%
  /* Grammar rules and actions follow.  */
 /* begin */
calclist:
    %empty
    |calclist exp EOL {printf("=%.10g\n",$2);}

exp     :term           { $$ = $1; }
        |exp ADD term   { $$ = $1 + $3;}
        |exp SUB term   { $$ = $1 - $3;}
        ;

term    :factor         { $$ = $1;}
        |term MUL factor { $$ = $1 * $3;}
        |term DIV factor { $$ = $1 / $3;}
        ;

factor  : NUM           {$$ = $1;}
        |LP exp RP      {$$ = $2;}
        |SUB factor     {$$ = -$2;}
        |ADD factor     {$$ = $2;}
        ;

 /* end */
%%
```

图 1-8 **语法规则**

task107:联合 Bison+flex 的模式，完成中缀表达式计算器。

语法规则如下图所示：

```
%%
calclist:
    %empty
    |calclist exp EOL {printf("=%.10g\n",$2);}

exp     :term
        |exp ADD exp {$$=$1+$3;}
        |exp SUB exp {$$=$1-$3;}
        |error {}
        ;

term    :pown         { $$ = $1;}
        |term MUL term { $$ = $1 * $3;}
        |term DIV term { $$ = $1 / $3;}
        ;

pown    :factor         {$$ = $1;}
        |pown EXPO pown   {$$ = pow($1, $3);}

factor  : NUM           {$$ = $1;}
        |LP exp RP      {$$ = $2;}
        |SUB factor     {$$ = -$2;}
        |ADD factor     {$$ = $2;}
        ;
%%
```

图 1-9 **语法规则**

（3）小结

通过实验一，我学习到了使用 Flex 和正则表达式进行简单的词法分析，使用 Bison 进行简单的语法分析，并将两者结合起来实现简单的计算器功能。

不足之处：词法分析部分存在较多的代码冗余。

1.2 MiniC 词法分析

（1）实验内容

Flex 与 Bison 工具后，我们将利用两种工具，逐步完成对 Mini-C 语言的结构分析。

（2）实验过程

task201：识别参考资料对应的单词。

实现方法：词法匹配规则如下图所示：

```
"int"         {flexout("TYPE","int");}
"float"       {flexout("TYPE","float");}
"char"        {flexout("TYPE","char");}
";"           {flexout("SEMI", ";");}
","           {flexout("COMMA", ",");}
"=="          {flexout("RELOP", "==");}
"+="          {flexout("PLUSASS", "+=");}
"-="          {flexout("MINUSASS", "-=");}
"="           {flexout("ASSIGNOP", "=");}
"++"          {flexout("PLUSPLUS", "++");}
"+"           {flexout("PLUS", "+");}
"--"          {flexout("MINUSMINUS", "--");
"-"           {flexout("MINUS", "-");}
"*"           {flexout("STAR", "*");}
"/"           {flexout("DIV", "/");}
"&"           {flexout("AND", "&");}
"|"           {flexout("OR", "|");}
"."           {flexout("DOT", ".");}
"!"           {flexout("NOT", "!");}
"("           {flexout("LP", "(");}
")"           {flexout("RP", ")");}
"["           {flexout("LB", "[");}
"]"           {flexout("RB", "]");}
"{"           {flexout("LC", "{");}
"}"           {flexout("RC", "}");}
```

图 1-10 匹配规则

task202：增加了对保留关键字的测试；能够识别简单浮点数，例如 1.2，1.05e5，八进制数、十六进制数等。同时，能做到一定程度的容错功能：识别非法八进制如 08、非法十六进制数字如 0xGF2。

实现方法：词法匹配规则如下图所示：

```
DIGIT ((\-)?([1-9][0-9]*))|0
ID [a-zA-Z_][a-zA-Z_0-9]*
FLOAT (\-)?{DIGIT}+\.{DIGIT}*(e{DIGIT})?

OCTAL 0[0-7]+
ILOCTAL 0[0-7]*[89\w\W]+[0-7]*

HEX 0(x|X)[A-Fa-f0-9]+
ILHEX 0(x|X)[A-Fa-f0-9]*[g-zG-Z]+[A-Fa-f0-9]*
```

图 1-11 匹配规则

task203：掌握规则顺序对词法分析的影响，修改词法规则，完成以下四个运算符的识别：++，--，+=，-=。

实现方法：词法匹配规则如下图所示：

```
"=="          {flexout("RELOP", "==");}
"+="          {flexout("PLUSASS", "+=");}
"-="          {flexout("MINUSASS", "-=");}
"="           {flexout("ASSIGNOP", "=");}
"++"          {flexout("PLUSPLUS", "++");}
"+"           {flexout("PLUS", "+");}
```

图 1-12 匹配规则

（3）小结

本次实验实现了对 minic 中的关键字、符号、整数、小数、八进制和十六进制数的分析，以及相应的错误检测机制。

不足之处：匹配规则存在冗余，鲁棒性不强，可能存在 bug。

1.3 MiniC 语法分析及语法树生成

（1）实验内容

学习词法、语法分析识别程序的编写方法，使用 flex，完成 Mini-C 语言词法分析。采用语法制导的方法，完成语法树的输出。

（2）实验过程

task301：了解 Bison 系统自动解决冲突的能力以及手动解决移进规约冲突与规约规约冲突，并动手消除状态中的移进规约冲突。

Bison 自动解决冲突：

```
cse@s-614:~/miniC/lab3/task301$ diff foo.output foo.output1
111,112d110
<     Conflict between rule 1 and token '+' resolved as reduce (%lef
t '+').
<
```

图 1-13 Bison 解决冲突

消除状态中的移进规约冲突：

```
//这个例子中有一些问题，请调试修改
%union
{
    int ival;
    char *sval;
} //**begin***//
%token <ival>NUM
%nterm <ival>exp
%left '-'
%left '+'
%%
exp:
    exp '+' exp
    |exp '-' exp
    |NUM
    ;
%% //**end**//
```

图 1-14 消除冲突

task302：输入 Mini-C 的源文件，找到最左规约序列，判断其语法是否正确，输出错误信息，并按最左规约的顺序，打印出规约过程中使用的非终结符名称。

实现方法：根据 Mini-C 语法，编写语法规则，输出非终结符名称。语法规则部分如下图：

```
program: ExtDefList {std::cout<<"Program"<<std::endl;} /*显示规则对应非终结符*/
        ;

ExtDefList: ExtDef ExtDefList {std::cout<<"ExtDefList"<<std::endl;} /* 全局变量定义序列
        | %empty {std::cout<<"ExtDefList"<<std::endl;}
        ;

ExtDef: Specifier ExtDecList SEMI {std::cout<<"ExtDef"<<std::endl;} /* 全局变量定义 */
        | Specifier SEMI {std::cout<<"ExtDef"<<std::endl;} /* 全局结构体定义 */
        | Specifier FunDec CompSt {std::cout<<"ExtDef"<<std::endl;} /* 函数式定义 */
```

图 1-15 语法规则

task303：利用 Bison -v 参数等解决编译过程中的移进规约冲突。

实现方法：添加优先级规则，如下图所示：

```
//由低到高的定义优先级
%right ASSIGNOP PLUSASS MINUSASS STARASS DIVASS // = += -= *= /=
%left OR         // ||
%left AND        // &&
%left RELOP      // < <= > >= == !=
%left PLUS MINUS    //
%left STAR DIV MOD  //  / * MOD
%right NOT PLUSPLUS MINUSMINUS //
%left DOT
%left LB RB LP RP
%nonassoc ELSE
```

图 1-16 优先级规则

测试结果如下图所示：

```
cse@s-614:~/miniC/lab3/task303$ ./minic taskcase/2.in
Specifier
FunDec
Specifier
VarDec
Dec
VarDec
Dec
VarDec
Dec
DecList
DecList
DecList
Def
DefList
DefList
```

图 1-17 测试结果

task304：对 Mini-C 的简单样例进行分析，按实验指导书要求，输出其对应的语法树。

实现方法：利用语法分析构建语法树，语法规则部分如下图所示：

```
program: ExtDefList {p=new NProgram($1);if($1) p->line=$1->line;} /*显示语法树*/
        ;
ExtDefList:
        %empty { $$=nullptr;}
        | ExtDef ExtDefList {$$=new NExtDefList(*$1,$2); $$->line=$1->line;}
        ;
ExtDef: Specifier ExtDecList SEMI {$$=new NExtDef(*$1,$2); $$->line=$1->line;}
        | Specifier SEMI {$$=new NExtDef(*$1); $$->line=$1->line;}
        | Specifier FunDec CompSt {$$=new NExtDef(*$1, $2, $3); $$->line=$1->line;}
        ;
ExtDecList: VarDec                    {$$=new NExtDecList(*$1, nullptr); $$->line=$1-
        | VarDec COMMA ExtDecList     {$$=new NExtDecList(*$1, $3); $$->line = $1->li
        ;
```

图 1-18 语法规则

测试结果如下图所示：

```
cse@s-614:~/miniC/lab3/task304$ ./minic taskcase/0.in
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
      FunDec (1)
        ID: main
        LP
        RP
      CompSt (2)
        LC
        DefList (3)
          Def (3)
            Specifier (3)
              TYPE: int
            DecList (3)
              Dec (3)
                VarDec (3)
                  ID: i
            SEMI
        StmtList (4)
          Stmt (4)
            RETURN
            Exp (4)
              INT: 0
```

图 1-19 测试结果

task305：完善语法树的输出。

实现方法：完善语法树的语法规则，并编写对应语法规则。部分语法规则如下图所示：

```
/*Specifiers*/
Specifier: TYPE { if ($1[0] == 'i') $$ = new NSpecifier(*(new std::string("int")));
                  else $$ = new NSpecifier(*(new std::string("float"))); $$->line = yylineno; }
        | StructSpecifier {$$ = new NStructSpSpecifier(*$1); $$->line = $1->line;}
        ;

StructSpecifier: STRUCT OptTag LC DefList RC    {$$ = new NStructSpecifier($2, $4); $$->line = $1;}
        | STRUCT Tag                            {$$ = new NStructSpecifier($2); $$->line = $1;}
        ;

OptTag: ID              {$$ = new NOptTag(*(new NIdentifier($1))); $$->line = yylineno - 1;}
        |%empty         {$$ = nullptr;}
        ;

Tag: ID {$$ = new NTag(*(new NIdentifier($1))); $$->line = yylineno;}
        ;
```

图 1-20 **语法规则**

测试结果如下图所示：

```
cse@s-614:~/miniC/lab3/task305$ ./minic testcase/0.in
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
      FunDec (1)
        ID: main
        LP
        RP
      CompSt (2)
        LC
        DefList (3)
          Def (3)
            Specifier (3)
              TYPE: int
            DecList (3)
              Dec (3)
                VarDec (3)
                  ID: i
            SEMI
        StmtList (4)
          Stmt (4)
            RETURN
            Exp (4)
              INT: 0
```

图 1-21 **测试结果**

（3）小结

本次实验中，完成了对 Mini-C 的语法树构建，能够输出完整的语法树，并对其行号的输出以及错误提示。

遇到的困难：在语法树的构建过程中，有几个特殊的类需要单独构建与判断。

不足之处：语法树中有两处的行号问题。

1.4 MiniC 语义分析及中间代码生成

（1）实验内容

编写程序，对输入的 Mini-C 源文件进行分析，输出对应的 IR 程序段。

（2）实验过程

task401: 手工编写 LLVM IR 程序实现多个输出与分支等功能。

实验一：输出 HUSTCSE\n。

实验二：输入单个字符，如果是 a 则输出 Y，否则输出 N。

实现方法：一、编写代码，先将字符对应 ASCII 码存入内存中，后将其取出放入参数内。最后 call putchar 函数输出。代码如下图所示：

二、编写代码，获取输入字符，将其与 a 字符比较，若结果相等则跳转到输出 Y，否则跳转到输出 N，测试结果如下图所示：

```
cse@s-614:~/miniC/lab4/task401$ lli task1.ll
HUSTCSE
```

图 1-22 **实验一测试结果**

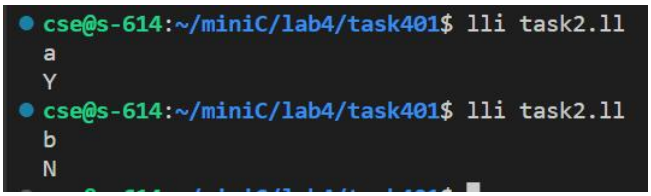

图 1-23 实验二测试结果

task402：使用 C++代码与 LLVM 提供的接口函数，动态生成 LLVM IR。

实现方法：实验一，获取字母对应 ASCII 常量，存入内存，加载常数，存入参数数组中，调用函数输出对应字符。实验二，创建基本块进行跳转，部分代码测试结果如下图所示：

图 1-24 实验一测试结果

图 1-25 实验二测试结果

task404: 编写编译器前端，分阶段（子任务）将 minic 源代码翻译到中间代码；并分析其中的语义错误。

实现方法：根据 IR 规则与语义分析，生成相应的代码与错误解析。测试结果如下图所示：

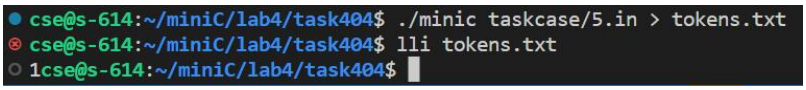

图 1-26 测试结果

task405：编写编译器前端，分阶段（子任务）将 minic 源代码翻译到中间代码；并分析其中的语义错误。

实现方法：根据 IR 规则与语义分析，生成相应的代码与错误解析，测试结果如下图所示：

图 1-27 测试结果

（3）小结

本实验实现了对 Mini-C 源文件的 LLVM IR 实现，完成了动态编译的目的。

遇到的困难：变量表的建立，维护和查找需要定义编写相关内容

不足之处：部分功能尚未实现。

1.5 MiniC 代码优化及目标代码生成

（1）实验内容

简单的使用 LLVM 代码优化框架及命令行，完成相关任务。

（2）实验过程

关卡相关的主要实现过程。

task501：调用 LLVM 中的优化函数，观察 IR 代码发生的变化。

实验结果：由于编写的问题，代码无变化。

task502：将文件 test.txt 中的 IR 进行优化，并最终能够产生可执行的二进制程序 test。

测试结果：

```
cse@s-614:~/miniC/lab5/task501$ opt --mem2reg test.txt -S > test.ll
cse@s-614:~/miniC/lab5/task501$ llc test.ll -o test.s
cse@s-614:~/miniC/lab5/task501$ clang test.s -o test
```

图 1-28 测试结果

task503：我们还可以构造自己的 PASS 实现输出函数名字及对应函数的基本块数量。

实现方法：重写 runOnFunction 函数，如下图所示：

```
bool runOnFunction(Function &F) override {
  outs().write_escaped(F.getName()) << ':';
  outs() << F.getBasicBlockList().size() << '\n';
  return false;
}
```

图 1-29 修改代码

测试结果如下图所示：

```
cse@s-614:~/miniC/lab5/task501$ make
clang++ -c countPass.cpp `llvm-config --cxxflags`
clang++ -shared -o countPass.so countPass.o `llvm-config --ldfl
opt -load ./countPass.so -CountPass test.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

calc:2
main:7
```

图 1-30 测试结果

（3）小结

通过本实验对 LLVM 本身的优化函数的调用，自己编写优化函数以及可执行文件的生成等。

不足之处：由于代码问题 LLVM 优化效果不明显。

二、实验心得

在本次实验中，我学习到了对于 Mini-C 语言编译器的基本实现，能够对基本的函数调用，运算，语句等具体的实现，复习了编译原理中所学的一些知识。

在实验一中我学习到了对 Flex,Bison 的基本使用，能够对文本中的词法进行简单的分析以及一些简单的语法分析。

在实验二中我学习到了更为复杂的词法分析，能够对八进制数，十六进制数以及小数等进行具体的识别。

在实验三中我学习到了 Bison 冲突的解决方法以及语法树的构建。

在实验四中我学习到了 LLVM IR 的编写以及如何依据语法树来动态生成 LLVM IR 文件。

在实验五中我学习到了 LLVM 中的一些代码优化函数，以及编写了自己的一些函数。

三、实验内容和过程的建议

本实验内容循序渐进，由简单到复杂来实现了一个编译器的构建。

优点：使用云平台，省略了本地配置环境的时间，同时使用 linux 无桌面端，流畅不卡顿。

使用云平台自测试，有一个统一的评分标准，方便学生去发现错误。

前三个实验难度适中，且说明较详细。

缺点：第三个实验 4，5。第四个实验 3，4，5 难度较大。且说明内容较少，需要学生自己补充的内容过多，示例较少，做起来力不从心，只能借鉴网络上的部分参考，才能实现相关内容。建议这部分加入一些提示，示例部分，同时减少需要自己补充的部分。

参考文献

[1] 许畅 等编著. 《编译原理实践与指导教程》.机械工业出版社

[2] Jobn Levine著，陆军 译. 《Flex与Bison》.东南大学出版社