華中科技大学

课程实验报告

题目：编译技术实验——编译器

课程名称：编译技术实验

专业班级：软件工程 1804

学　　号：U201817070

姓　　名：刘书哲

指导教师：胡雯蔷

报告日期：2020 年 11 月 23 日

软件学院

概述

本次实验是构造一个高级语言的子集的编译器，目标代码是LLVM IR 中间语言。

这里实现了一个精简版 C语言编译期，只有基础的简单的语法。实验的任务主要是通过对简单编译器的完整实现，加深课程中关键算法的理解，提高学生系统软件研发技术。

系统描述

系统流程

源代码 => [Flex/Bison 生成的文法分析器] => AST => [语义分析器] => Code Node => [LLVM API] => LLVM IR

自定义语言描述

是 C 语言的一个简单的子集，简称为 Mini-C

文法描述

详见 `lex.l` 和 `parser.y` ;

```
%type <ptr> program ExtDefList ExtDef Specifier ExtDecList FuncDec CompSt
VarList VarDec ParamDec Stmt StmList DefList Def DecList Dec Exp Args

%token <type_int> INT
%token <type_id> ID RELOP TYPE
%token <type_float> FLOAT

%token DPLUS LP RP LC RC SEMI COMMA
%token PLUS MINUS STAR DIV MOD ASSIGNOP AND OR NOT IF ELSE WHILE RETURN
FOR SWITCH CASE COLON DEFAULT CONTINUE BREAK
%token EXT_DEF_LIST EXT_VAR_DEF FUNC_DEF FUNC_DEC EXT_DEC_LIST PARAM_LIST
PARAM_DEC VAR_DEF DEC_LIST DEF_LIST COMP_STM STM_LIST EXP_STMT IF_THEN
IF_THEN_ELSE
%token FUNC_CALL ARGS FUNCTION PARAM ARG CALL LABEL GOTO JLT JLE JGT JGE
EQ NEQ

%left ASSIGNOP
%left OR
%left AND
%left RELOP
%left PLUS MINUS
%left STAR DIV
%left MOD
%right UMINUS NOT DPLUS

%nonassoc LOWER_THEN_ELSE
%nonassoc ELSE

%%
```

```
program: ExtDefList { print_ast_node($1,0); entrypoint($1);};
ExtDefList: {$$=nullptr;}
| ExtDef ExtDefList {$$=make_node(EXT_DEF_LIST,yylineno,{$1,$2});};
ExtDef: Specifier ExtDecList SEMI {$$=make_node(EXT_VAR_DEF,yylineno,
{$1,$2});}
|Specifier FuncDec CompSt {$$=make_node(FUNC_DEF,yylineno,{$1,$2,$3});}
| error SEMI {$$=nullptr;};
Specifier: TYPE {$$=make_node(TYPE,yylineno);$$->data = string($1);$$-
>type=(string($1) == "float")?FLOAT:INT;} ;
ExtDecList: VarDec {$$=$1;}
| VarDec COMMA ExtDecList {$$=make_node(EXT_DEC_LIST,yylineno,{$1,$3});};
VarDec: ID {$$=make_node(ID,yylineno);$$->data = $1;};
FuncDec: ID LP VarList RP {$$=make_node(FUNC_DEC,yylineno,{$3});$$->data
= $1;}
|ID LP RP {$$=make_node(FUNC_DEC,yylineno);$$->data = $1;$$-
>ptr[0]=nullptr;};
VarList: ParamDec {$$=make_node(PARAM_LIST,yylineno,{$1});}
| ParamDec COMMA VarList {$$=make_node(PARAM_LIST,yylineno,{$1,$3});};
ParamDec: Specifier VarDec {$$=make_node(PARAM_DEC,yylineno,{$1,$2});};
CompSt: LC DefList StmList RC {$$=make_node(COMP_STM,yylineno,{$2,$3});};
StmList: {$$=nullptr; }
| Stmt StmList {$$=make_node(STM_LIST,yylineno,{$1,$2});};
Stmt: Exp SEMI {$$=make_node(EXP_STMT,yylineno,{$1});}
| CompSt {$$=$1;}
| RETURN Exp SEMI {$$=make_node(RETURN,yylineno,{$2});}
| IF LP Exp RP Stmt %prec LOWER_THEN_ELSE {$$=make_node(IF_THEN,yylineno,
{$3,$5});}
| IF LP Exp RP Stmt ELSE Stmt {$$=make_node(IF_THEN_ELSE,yylineno,
{$3,$5,$7});}
| WHILE LP Exp RP Stmt {$$=make_node(WHILE,yylineno,{$3,$5});}
| CONTINUE SEMI {$$=make_node(CONTINUE,yylineno);}
| BREAK SEMI {$$=make_node(BREAK,yylineno);};
DefList: {$$=nullptr; }
| Def DefList {$$=make_node(DEF_LIST,yylineno,{$1,$2});}
| error SEMI {$$=nullptr;};
Def: Specifier DecList SEMI {$$=make_node(VAR_DEF,yylineno,{$1,$2});};
DecList: Dec {$$=make_node(DEC_LIST,yylineno,{$1});}
| Dec COMMA DecList {$$=make_node(DEC_LIST,yylineno,{$1,$3});};
Dec: VarDec {$$=$1;}
| VarDec ASSIGNOP Exp {$$=make_node(ASSIGNOP,yylineno,{$1,$3});};
Exp: Exp ASSIGNOP Exp {$$=make_node(ASSIGNOP,yylineno,{$1,$3});}
| Exp AND Exp {$$=make_node(AND,yylineno,{$1,$3});}
| Exp OR Exp {$$=make_node(OR,yylineno,{$1,$3});}
| Exp RELOP Exp {$$=make_node(RELOP,yylineno,{$1,$3});$$->data = $2;}
| Exp PLUS Exp {$$=make_node(PLUS,yylineno,{$1,$3});}
| Exp MINUS Exp {$$=make_node(MINUS,yylineno,{$1,$3});}
| Exp STAR Exp {$$=make_node(STAR,yylineno,{$1,$3});}
| Exp MOD Exp {$$=make_node(MOD,yylineno,{$1,$3});}
| Exp DIV Exp {$$=make_node(DIV,yylineno,{$1,$3});}
| LP Exp RP {$$=$2;}
| MINUS Exp %prec UMINUS {$$=make_node(UMINUS,yylineno,{$2});}
| NOT Exp {$$=make_node(NOT,yylineno,{$2});}
| DPLUS Exp {$$=make_node(DPLUS,yylineno,{$2});}
| Exp DPLUS {$$=make_node(DPLUS,yylineno,{$1});}
| ID LP Args RP {$$=make_node(FUNC_CALL,yylineno,{$3});$$->data = $1;}
| ID LP RP {$$=make_node(FUNC_CALL,yylineno);$$->data = $1;}
| ID {$$=make_node(ID,yylineno);$$->data = $1;}
| INT {$$=make_node(INT,yylineno);$$->data=$1;$$->type=INT;}
```

```
| FLOAT {$$=make_node(FLOAT,yylineno);$$->data=$1;$$->type=FLOAT;};
Args: Exp COMMA Args {$$=make_node(ARGS,yylineno,{$1,$3});}
| Exp {$$=make_node(ARGS,yylineno,{$1});};
%%
```

符号表定义

最终提交的源代码中，使用了 `std::vector` 作为了符号表的数据结构；根据编译的推进顺序将符号按照顺序加入表中，查找时始终倒序遍历，就可以查找到最新的符号（也就是需要查找的那个符号）

```
enum SymbolFlag {
    fFunc = 'F',
    fVar = 'V',
    fParam = 'P',
    fTemp = 'T',
    fNull = '\0'
};

struct Symbol {
    string name;
    int level, type;
    int paramNum;
    string alias;
    SymbolFlag flag;
    int idx;

    Symbol();
};

using SymbolTable = symbol_table_t<Symbol>;
```

上面的代码中，使用 `SymbolFlag` 枚举符号表中的符号可能的类型；

中间代码定义

中间代码整体的结构是一棵有分支的链（树）；链的顺序代表着表达式的顺序，节点的分支代表归约的结果。

```
struct Code {
    int kind;
    vector<shared_ptr<Code>> data;
    Opt op1, op2, res;
    shared_ptr<Code> prev, next;

    Code();
};
```

生成中间代码的目的是为了下一步交给 LLVM 生成 IR 码。

目标代码定义

本实验生成的目标代码是 LLVM IR，使用其 API 构造目标代码。

设计和实现

文法分析器

使用 Flex 和 Bison 实现；通过书写特定规则的文件来生成对应的分析器程序，并调用相应的函数生成 AST；

符号表管理

符号表的基本操作使用 C++ 的 `std::vector` 维护；使用一个栈维护符号表坐标区间的符号所属的作用域；当离开一个作用域的时候，利用维护的信息 `resize` 快速清除不需要的符号；

语义分析

在文法转换器生成的 AST 上进行 DFS 搜索，并且同时维护符号表，并且进行一致性检查；当检查不通过的时候报错；与此同时将分析的结果存入代码节点，以方便下一步目标代码的生成。

开发环境

- WSL Debian 10 (based on WSL 2, Windows 10 Pro 20H2 19042.630)
- Jetbrains CLion 2020.2.1 Build #CL-202.6948.80
- GNU GCC version 8.3.0 (Debian 8.3.0-6)
- GNU Bison 3.3.2
- LLVM 10.0.1 (on Debian 10)
- flex 2.6.4
- CMake version 3.13.4

目标代码

根据语义分析过程中生成的代码节点，调用 LLVM IR 提供的 API 生成 LLVM IR Code；因为 LLVM IR Code 受到 LLVM 的编译后端的支持，所以通过它生成可以运行在任何主流平台上机器码，从而执行。

测试和评价

测试用例

在 `test/` 目录下有可以正确通过编译的代码；

下面的测试样例会使得编译器报错：

```
int main() {
    print_int(b);
    return 0;
}
```

产生的错误信息是：`Line 2, Message: b Variable not declared.`

优点

- 通过 `std::exception` 、 `std::shared_ptr` 等 C++17 的新 API，非常现代
- 使用了 LLVM 作为编译器的后端，直接生成 LLVM IR 中间码，具有较强的泛用性
- 基于 CMake 编写了全自动环境安装、编译、构建、执行的脚本，方便操作人员调试

缺点

- 有很多的功能没有实现：比如一元运算符和数组访问
- 符号表处理不够优雅，可以使用更加优秀的算法或者数据结构来提高效率
- 出错处理不够完善全面，很多时候还需要程序员进行调试

附录

源代码地址： ma-hunter/toy-compiler (github.com) https://github.com/ma-hunter/toy-compiler

执行：使用 CLion 自动 Configure 或者进入仓库的 `code/` 目录下运行 `bash start.sh`