MiniJava 编译器前端 report

刘卓珉-15307130223

分组情况

个人一组完成

使用工具

bison flex

使用方式

详见 README.md

完成功能

1、 完整的词法、语法解析

2、 词法错误和语法错误的判断，详见 parser.y 和 lexer.l

词法错误判断部分 lexer.l:

```
.                       { fprintf(stderr, "Unkown word."); }
```

语法错误判断部分 parser.y:

```
void yyerror(char *s) {
    fprintf(stderr, "line %d: %s \n", yylineno, s);
}
```

3、 部分语义错误判断，如变量方法是否定义，类型错误等，详见 node.c

语义错误判断:

```
// check semantics error
int class_size = 0;
struct Node **classes = NULL;

void resolve_all_class(struct Node *goal) {
    int class_size = 1;
    struct Node *r = goal->children[1];
    if (r != NULL) {
        r = r->children[0];
        class_size++;
        while (r->children_size < 2) {
            r = r->children[0];
            class_size++;
        }
    }
    classes = (struct Node**)malloc(sizeof(struct Node*) * class_size);
    r = goal->children[1];
    classes[0] = goal->children[0];
    if (r != NULL) {
        r = r->children[0];
        int cnt = 1;
        while (r->children_size < 2) {
            classes[cnt] = r->children[1];
            cnt++;
            r = r->children[0];
        }
        classes[cnt] = r->children[0];
    }
}

int check_all_type(struct Node *r) {
    if (strcmp(r->val, "Type") == 0 && r->children_size == 1) {
        struct Node *id = r->children[0];
        if (strcmp(id->val, "int") == 0 || strcmp(id->val, "intean") == 0) {
            return 1;
        }
        int i;
        for (i = 0; i < class_size; i++) {
            if (strcmp(id->val, classes[i]->children[2]->val) == 0) {
                return 1;
            }
        }
        fprintf(stderr, "line %d: no type called %s\n", id->line, id->val);
        return 0;
    }
    int i;
    for (i = 0; i < r->children_size; i++) {
        if (!check_all_type(r->children[i])) return 0;
    }
    return 1;
}
```

```
int check_all_var_in_method(struct Node *statements,
                            struct Node *var_declarations_in_class,
                            struct Node *type_identifiers,
                            struct Node *var_declarations) {
    if (statements == NULL) return 1;
    if (statements->children[0]->val[0] == 'i' && statements->children[0]->val[1] == 'd') {
        struct Node *v = var_declarations_in_class;
        while (v != NULL) {
            struct Node *id = v->children[1]->children[1];
            if (strcmp(id->val, statements->children[0]->val) == 0)
                return 1;
            v = v->children[0];
        }
        if (type_identifiers != NULL) {
            v = type_identifiers->children[0];
            while (v->children_size == 2) {
                struct Node *id = v->children[2]->children[1];
                if (strcmp(id->val, statements->children[0]->val) == 0)
                    return 1;
                v = v->children[0];
            }
            struct Node *id = v->children[0]->children[1];
            if (strcmp(id->val, statements->children[0]->val) == 0)
                return 1;
        }
        v = var_declarations;
        while (v != NULL) {
            struct Node *id = v->children[1]->children[1];
            if (strcmp(id->val, statements->children[0]->val) == 0)
                return 1;
            v = v->children[0];
        }
        fprintf(stderr, "line %d: no type called %s\n",
                statements->children[0]->line,
                statements->children[0]->val);
        return 0;
    }
    int i = 0;
    for (; i < statements->children_size; i++) {
        if (!check_all_var_in_method(statements->children[i],
                                     var_declarations_in_class,
                                     type_identifiers,
                                     var_declarations)) {
            return 0;
        }
    }
    return 1;
}
```

```c
int check_all_var(struct Node *class) {
    struct Node *var_declarations_in_class = class->children[4];
    struct Node *method = class->children[5];
    if (method != NULL) {
        while (method->children_size == 2) {
            if (!check_all_var_in_method(method->children[1]->children[8],
                                         var_declarations_in_class,
                                         method->children[1]->children[4],
                                         method->children[1]->children[7]))
                return 0;
            method = method->children[0];
        }
        if (!check_all_var_in_method(method->children[0]->children[8],
                                     var_declarations_in_class,
                                     method->children[0]->children[4],
                                     method->children[0]->children[7]))
            return 0;
    }
    return 1;
}

int check(struct Node *goal) {
    resolve_all_class(goal);
    if (!check_all_type(goal)) return 0;
    int i;
    for (i = 1; i < class_size; i++) {
        if (!check_all_var(classes[i])) return 0;
    }
    return 1;
}

```

样例文件抽象语法树打印

截图详见 README.md

工具使用与源代码心得

1、 为什么使用 Bison+Flex 而不是 ANTLR

Bison/Flex 是新一代 yacc 和 lex，使用的是 LR 算法，而 ANTLR 使用的是 LL 算法，这两种方式各有优劣，至于我为什么选择 Bison 和 Flex，是因为我比较熟悉这两个工具，我曾在做 SQL engine 的时候用过 yacc 的 go 语言版本，所以就选择 Bison 和 Flex。

2、 碰到的 shift/reduce 错误与解决

开发中遇到了两种 shift/reduce 错误，第一种是因为左括号没有定义为运算符，导致中括号在某些语法表达里面存在 shift/reduce 的冲突，解决方式就是将其定义为运算符。第二个错误是，由于 yacc 中的可空 list 需要额外的非终止运算符定义，所以 varDeclarationList 的最初定义版本存在冲突，于是我换了另一种更简洁的方式定义，将冲突 resolve 掉了。

3、 额外的功能

编译器前端额外功能主要有两种：第一个是对于常数的解析我分为了二进制、八进制、十进制和十六进制，使编译器更完整，第二个是抽象语法树的打印，对于每个非终止符和终止符，都建立了对应节点，最终保存了树形的 AST 结构，同时语义错误的检测也是基于这个 AST 结构。