中国地质大学（北京）

编译原理课程设计报告

指导教师：耿明芹

姓　　名：王俊博

学　　号：1004211127

邮　　箱：junbowang@email.cugb.edu.cn

日　　期：2024.05.28

目录

1. 编译程序概述

编译程序是将高级编程语言代码转换为低级机器代码或中间代码的工具。它通过一系列步骤，将源代码解析并生成可以在计算机上执行的程序。编译器的主要目标是提高代码的执行效率和降低内存占用。

1.1 功能

- **词法分析**：将源代码转换为一系列词法单元（tokens），如关键字、标识符、常量、操作符和分隔符。
- **语法分析**：根据预定义的语法规则，将词法单元序列组织成语法树。
- **语义分析**：确保程序的语义正确性，进行类型检查和作用域管理。
- **中间代码生成**：将语法树转换为中间代码，便于后续优化和生成目标代码。
- **代码优化**：通过各种技术优化中间代码，提升代码执行效率。
- 目标代码生成：将中间代码转换为目标机器语言或汇编代码。
- 代码生成：生成目标平台上的可执行代码。

1.2编译程序结构

- **词法分析器（Lexer)**：负责将源代码分解为基本的词法单元。
- **语法分析器（Parser)**：将词法单元组织成语法树，检查语法正确性。
- **语义分析器**：进行语义检查，确保程序的逻辑正确性。
- **语义分析器**：生成与平台无关的中间代码。
- 代码优化器：优化中间代码，提高执行效率。
- 目标代码生成器：将中间代码转换为目标机器代码。
- 符号表：管理变量和函数的信息，包括名称、类型、作用域等。

2. 词法分析器

2.1 功能

- 根据预定义的关键字、常量、操作符和分隔符，对输入的字符串进行分词，并返回各个词法单元的类型和属性。
- 够计算输入字符串中的词法单元数量。

2.2 实现

词法分析器根据输入的字符串逐字符遍历，根据字符的类型判断词法单元。对于关键字、常量、操作符和分隔符，使用字典进行存储和查找，以实现快速的词法单元类型判断。使用递归或迭代方式实现指针的移动和词法单元的识别。
在识别过程中，对于连续的数字字符，将其视为一个整体的常量；对于连续的字母字符和下划线，将其视为一个标识符。对于未识别的字符，抛出 ValueError 异常。

初始化方法 (__init__):

定义了词法分析器的基本属性，包括输入字符串 (string) 和当前指针位置 (pointer)。
构建了四个字典，分别存储关键字 (keywords)、常量 (constants)、操作符 (operators) 和分隔符 (delimiters)，并为每个词法单元分配了唯一的标识符。

count_tokens 方法:

用于计算输入字符串中的词法单元数量。
通过遍历输入字符串的每个字符，根据字符的类型（数字、字母、运算符、分隔符等）进行识别，从而计数。

is_identifier 方法:

用于判断给定的字符串是否为标识符。
当字符串不是关键字时，被认为是标识符，返回一个特定的标识符代表标识符类型。

get_next_token 方法:

从输入字符串中获取下一个词法单元。
根据当前指针位置，判断当前字符的类型并返回相应的词法单元，同时更新指针位置。

analyze 方法:

调用 get_next_token 方法获取下一个词法单元，直到遍历完整个输入字符串。
如果指针已经超出字符串长度，返回 None。

analyze_file 方法:

从文件中读取内容，并调用 analyze 方法进行词法分析。
在文件不存在时，输出错误信息。

2.3 测试

输入：while(a[i])
　　b[i,j]=10;
输出：

```
Token num:16
第 1 次调用词法分析器(20, '-')
第 2 次调用词法分析器(81, '-')
```

第 3 次调用词法分析器(111, 'a')
第 4 次调用词法分析器(88, '-')
第 5 次调用词法分析器(111, 'i')
第 6 次调用词法分析器(89, '-')
第 7 次调用词法分析器(82, '-')
第 8 次调用词法分析器(111, 'b')
第 9 次调用词法分析器(88, '-')
第 10 次调用词法分析器(111, 'i')
第 11 次调用词法分析器(83, '-')
第 12 次调用词法分析器(111, 'j')
第 13 次调用词法分析器(89, '-')
第 14 次调用词法分析器(46, '-')
第 15 次调用词法分析器(100, '整数')
第 16 次调用词法分析器(84, '-')

3. 语法分析器

3.1 功能

按照预定义的语法规则对其进行分析，并执行相应的语义动作。具体来说，Parser 实现了对布尔表达式的解析，包括算术表达式、关系运算和逻辑运算等。

3.2 实现

初始化方法 (__init__):
在初始化时，Parser 接受一个 Lexer 对象作为参数，从 Lexer 中获取 token 流。
eat 方法:
eat 方法用于消耗当前 token，检查是否与预期的 token 类型匹配，如果匹配则向前移动一个 token，否则引发异常。
print_step 方法:
print_step 方法用于打印分析过程中的产生式，便于调试和理解分析过程。
factor 方法:
factor 方法用于解析表达式的最小单元，包括数字、标识符和括号中的表达式。
unary 方法:
unary 方法用于处理一元操作符，目前仅支持因子(factor)。
term 方法:
term 方法解析算术表达式的项，支持乘法和除法运算。
expression 方法:
expression 方法解析算术表达式，支持加法和减法运算。
rop_expr 方法:

rop_expr 方法解析关系表达式，支持<、<=、>、>=运算符。
rel 方法:
rel 方法解析关系表达式，用于处理关系表达式与关系表达式之间的关系。
equality 方法:
equality 方法解析等式表达式，支持==和!=运算符。
bool_expr 方法:
bool_expr 方法解析布尔表达式，是整个解析过程的入口。
parse 方法:
parse 方法是解析器的入口，开始解析布尔表达式

3.3 测试

输入：

```
if(flag==1)
    while(i<=100)
        i=c[i]*d;
else
    z[i,j]=a+k;
s=t;
```

输出：

(1) stmts ⟶ stmt rest0
(2) stmt ⟶ if (bool) stmt else stmt
(3) bool ⟶ equality
(4) equality ⟶ rel rest4
(5) rel ⟶ expr rop_expr
(6) expr ⟶ term rest5
(7) term ⟶ unary rest6
(8) unary ⟶ factor
(9) factor ⟶ loc
(10) loc ⟶ id resta
(11) resta ⟶ ε
(12) rest6 ⟶ ε
(13) rest5 ⟶ ε
(14) rop_expr ⟶ ε
(15) rest4 ⟶ == rel rest4
(16) rel ⟶ expr rop_expr
(17) expr ⟶ term rest5
(18) term ⟶ unary rest6
(19) unary ⟶ factor
(20) factor ⟶ num
(21) rest6 ⟶ ε
(22) rest5 ⟶ ε

(23) rop_expr ⟶ ε
(24) rest4 ⟶ ε
(25) stmt ⟶ while (bool) stmt
(26) bool ⟶ equality
(27) equality ⟶ rel rest4
(28) rel ⟶ expr rop_expr
(29) expr ⟶ term rest5
(30) term ⟶ unary rest6
(31) unary ⟶ factor
(32) factor ⟶ loc
(33) loc ⟶ id resta
(34) resta ⟶ ε
(35) rest6 ⟶ ε
(36) rest5 ⟶ ε
(37) rop_expr ⟶ <= expr
(38) expr ⟶ term rest5
(39) term ⟶ unary rest6
(40) unary ⟶ factor
(41) factor ⟶ num
(42) rest6 ⟶ ε
(43) rest5 ⟶ ε
(44) rest4 ⟶ ε
(45) stmt ⟶ loc = expr;
(46) loc ⟶ id resta
(47) resta ⟶ ε
(48) expr ⟶ term rest5
(49) term ⟶ unary rest6
(50) unary ⟶ factor
(51) factor ⟶ loc
(52) loc ⟶ id resta
(53) resta ⟶ [elist]
(54) elist ⟶ expr rest1
(55) expr ⟶ term rest5
(56) term ⟶ unary rest6
(57) unary ⟶ factor
(58) factor ⟶ loc
(59) loc ⟶ id resta
(60) resta ⟶ ε
(61) rest6 ⟶ ε
(62) rest5 ⟶ ε
(63) rest1 ⟶ ε
(64) rest6 ⟶ * unary rest6

(65) unary ⟶ factor
(66) factor ⟶ loc
(67) loc ⟶ id resta
(68) resta ⟶ ε
(69) rest6 ⟶ ε
(70) rest5 ⟶ ε
(71) stmt ⟶ loc = expr;
(72) loc ⟶ id resta
(73) resta ⟶ [elist]
(74) elist ⟶ expr rest1
(75) expr ⟶ term rest5
(76) term ⟶ unary rest6
(77) unary ⟶ factor
(78) factor ⟶ loc
(79) loc ⟶ id resta
(80) resta ⟶ ε
(81) rest6 ⟶ ε
(82) rest5 ⟶ ε
(83) rest1 ⟶ , expr rest1
(84) expr ⟶ term rest5
(85) term ⟶ unary rest6
(86) unary ⟶ factor
(87) factor ⟶ loc
(88) loc ⟶ id resta
(89) resta ⟶ ε
(90) rest6 ⟶ ε
(91) rest5 ⟶ ε
(92) rest1 ⟶ ε
(93) expr ⟶ term rest5
(94) term ⟶ unary rest6
(95) unary ⟶ factor
(96) factor ⟶ loc
(97) loc ⟶ id resta
(98) resta ⟶ ε
(99) rest6 ⟶ ε
(100) rest5 ⟶ + term rest5
(101) term ⟶ unary rest6
(102) unary ⟶ factor
(103) factor ⟶ loc
(104) loc ⟶ id resta
(105) resta ⟶ ε
(106) rest6 ⟶ ε

(107) rest5 ⟶ ε
(108) rest0 ⟶ stmt rest0
(109) stmt ⟶ loc = expr;
(110) loc ⟶ id resta
(111) resta ⟶ ε
(112) expr ⟶ term rest5
(113) term ⟶ unary rest6
(114) unary ⟶ factor
(115) factor ⟶ loc
(116) loc ⟶ id resta
(117) resta ⟶ ε
(118) rest6 ⟶ ε
(119) rest5 ⟶ ε
(120) rest0 ⟶ ε

4. 语义分析和中间代码生成器

4.1 功能

- 解析赋值语句： 识别形如 a=6/b+5*c-d;的赋值语句，包括支持数组元素的赋值操作。
- 解析算术表达式：识别并解析算术表达式，包括支持加法、减法、乘法和除法等基本运算符。
- 解析布尔表达式： 能够识别并解析布尔表达式，包括支持比较运算符（如等于、不等于、大于、小于等）。
- 解析条件语句： 能够识别并解析 if-else 条件语句，生成相应的控制流跳转四元式。
- 解析循环语句： 能够识别并解析 while 循环语句，生成相应的控制流跳转四元式。
- 生成四元式： 在解析过程中，能够生成相应的四元式表示中间代码，以便后续进行优化和目标代码生成。
- 符号表管理： 使用符号表来管理变量和数组的信息，包括名称、维数、地址等

4.2 实现

Makelist 方法：创建一个包含单个四元式索引的列表，用于控制流跳转指令的回填。
backpatch 方法：回填函数，将列表中的所有四元式的跳转目标设置为指定的目标位置。

merge 方法：合并两个四元式列表，用于处理控制流的多个出口。
emit 方法：生成四元式并将其添加到四元式列表中，包含操作符、两个操作数和结果。
newtemp 方法：生成新的临时变量，用于存储中间计算结果。
Truelist（真列表）：在生成的四元式（quadruples）中，当条件为真时，程序应该跳转到特定目标指令的位置索引（或占位符）的列表
Falselist（假列表）：在生成的四元式中，当条件为假时，程序应该跳转到特定目标指令的位置索引（或占位符）的列表。
在评估布尔表达式时，如果条件为假，控制应该跳转到 falselist 中指定的位置

4.2.1 赋值语句的翻译

在处理赋值语句时， **stmt 方法**会进行以下步骤：
识别赋值语句：首先，检查当前 token 是否是标识符（即变量名）。如果是，表示这是一条赋值语句。
解析左值：调用 loc 函数解析赋值号左边的变量或数组元素，获取它们的地址。
解析右值：调用 expression 函数解析赋值号右边的表达式，计算其值。
生成赋值指令：根据左值和右值的解析结果，生成相应的中间代码。如果左值是变量，则生成简单的赋值指令；如果是数组元素，则生成数组赋值指令

4.2.2 数组的翻译

loc 方法处理数组的访问和操作：
获取标识符：读取当前 token，获取变量名或数组名。
符号表查询：在符号表中查找该标识符的条目，如果不存在则创建一个新的条目。
处理数组下标：调用 resta 函数处理数组的下标部分，计算具体元素的地址。
返回结果：返回变量或数组元素的地址及其偏移量。
resta 方法处理数组下标部分：
检测数组下标：检查当前 token 是否为 '['，如果是，则表示这是一个数组访问。
解析下标表达式：调用 **elist 方法**解析数组的下标列表，并计算偏移量。
生成地址计算指令：根据数组的基地址和偏移量，生成计算数组元素地址的中间代码。
elist 方法解析数组的下标列表：
解析第一个下标：调用 **expression 方法**解析第一个下标表达式。
处理剩余下标：调用 **rest1 方法**处理后续的下标表达式，并计算最终的偏移量。
rest1 方法处理数组的剩余下标部分：

检测是否有更多下标：检查当前 token 是否为 ','，如果是，则继续解析下一个下标表达式。
计算多维数组的偏移量：根据当前下标和之前解析的结果，生成相应的地

址计算指令。

4.2.3 布尔表达式的翻译

bool_expr 方法负责处理布尔表达式，主要调用 **equality 方法**来进行实际解析和中间代码生成。

equality 方法 处理等式表达式：

解析关系表达式：调用 rel 函数解析关系表达式，并获取其 truelist 和 falselist

处理后续部分：调用 rest4 函数处理后续的等式表达式，合并 truelist 和 falselist

rel 方法处理关系表达式：

解析算术表达式：调用 **expression 方法**解析表达式的左操作数。

处理关系运算符：调用 **rop_expr 方法**处理关系运算符及右操作数，生成比较指令并设置 truelist 和 falselist

rop_expr 方法处理关系运算符及其右操作数：

识别运算符：检查当前 token 是否为关系运算符（ '>', '<', '>=', '<=')。

解析右操作数：调用 expression 函数解析右操作数。

生成跳转指令：根据关系运算符生成相应的条件跳转指令，设置**方法**

4.2.4 控制语句的翻译

If 语句：

解析条件：调用 **bool_expr 方法** 解析 if 条件表达式，获取 truelist 和 falselist。

解析 if 分支：解析 if 语句块，并记录其结束位置。

生成跳转指令：在 if 分支之后生成无条件跳转指令，以跳过 else 分支。

解析 else 分支：解析 else 语句块。

回填 truelist 和 falselist：将条件表达式的 truelist 指向 if 分支，falselist 指向 else 分支。

While 语句：

记录循环开始位置：记录 while 语句开始的位置，用于循环的跳转。

解析条件：调用 **bool_expr 方法**解析 while 条件表达式，获取 truelist 和 falselist。

解析循环体：解析 while 循环体，并在循环体结束后添加跳转到循环开始位置的指令。

回填 truelist 和 falselist：将循环体的结束位置指向条件判断的开始位置，条件为真时进入循环体，条件为假时跳出循环。

4.3测试

输入：

```
    while(a<b)
        if(c)
            while(d>e)
                x1=y1;
        else
            x2=y2;
    x3[k]=y3[i,j];
```

输出：

```
0: ['j<', 'a', 'b', 2]
1: ['j', '-', '-', 11]
2: ['jnz', 'c', '-', 4]
3: ['j', '-', '-', 9]
4: ['j>', 'd', 'e', 6]
5: ['j', '-', '-', 0]
6: ['=', 'y1', '-', 'x1']
7: ['j', '-', '-', 4]
8: ['j', '-', '-', 0]
9: ['=', 'y2', '-', 'x2']
10: ['j', '-', '-', 0]
11: ['-', 'x3', 'C', 't1']
12: ['*', 'k', 'w', 't2']
13: ['*', 'i', 'n2', 't3']
14: ['+', 't3', 'j', 't3']
15: ['-', 'y3', 'C', 't4']
16: ['*', 't3', 'w', 't5']
17: ['=[]', 't4[t5]', '-', 't6']
18: ['[]=', 't6', '-', 't1[t2]']
```

5. 总结

在本次课程中，我们成功实现了编译程序的几个关键部分，包括词法分析器（Lexer）、语法分析器（Parser）、语义分析器以及中间代码生成器。这些组件各自完成了特定的功能，共同构成了一个简单但完整的编译流程。通过详细的功能设计和测试，我们验证了编译器各个组件的正确性和有效性。这为后续的代码优化和目标代码生成打下了坚实的基础。未来的工作将集中在进一步优化中间代码、生成高效的目标代码以及完善更多的编译功能，以支持更复杂的编程语言特性。

附录 源代码

```
class Lexer:
    def __init__(self):
        self.string = ""
        self.pointer = 0
        self.keywords = {
            "int": (5, "-"),
            "else": (15, "-"),
            "if": (17, "-"),
            "while": (20, "-")
        }

        self.constants = {
            "整数": (100, "整数")
        }

        self.operators = {
            "+": (41, "-"),
            "-": (42, "-"),
            "*": (43, "-"),
            "/": (44, "-"),
            "%": (45, "-"),
            "=": (46, "-"),
            ">": (47, "-"),
            ">=": (48, "-"),
            "<": (49, "-"),
            "<=": (50, "-"),
            "==": (51, "-"),
            "!=": (52, "-"),
            "&&": (53, "-"),
            "||": (54, "-"),
            "!": (55, "-"),
            "++": (56, "-"),
            "--": (57, "-")
        }

        self.delimiters = {
            "(": (81, "-"),
            ")": (82, "-"),
            ",": (83, "-"),
```

```
            ";": (84, "-"),
            "{": (86, "-"),
            "}": (87, "-"),
            "[": (88, "-"),
            "]": (89, "-")
        }
    def get_operator_symbol(self, op):
        for symbol, (token_type, _) in self.operators.items():
            if token_type == op:
                return symbol
        return None
    def count_tokens(self):
        tokens = []
        current_token = ""
        i = 0
        while i < len(self.string):
            if self.string[i].isdigit():
                current_token += self.string[i]
                i += 1
                while i < len(self.string) and self.string[i].isdigit():
                    current_token += self.string[i]
                    i += 1
                tokens.append((100, current_token))
                current_token = ""
            elif self.string[i].isalnum() or self.string[i] == '_':
                current_token += self.string[i]
                i += 1
                while i < len(self.string) and (self.string[i].isalnum() or self.string[i] ==
'_'):
                    current_token += self.string[i]
                    i += 1
                if current_token in self.keywords:
                    tokens.append(self.keywords[current_token])
                else:
                    tokens.append(self.is_identifier(current_token))
                current_token = ""
            elif self.string[i] in self.operators or self.string[i:i+2] in self.operators:
                if self.string[i:i+2] in self.operators:
                    tokens.append(self.operators[self.string[i:i+2]])
                    i += 2
                else:
                    tokens.append(self.operators[self.string[i]])
```

```
                i += 1
            elif self.string[i] in self.delimiters:
                tokens.append(self.delimiters[self.string[i]])
                i += 1
            elif self.string[i].isspace():
                i += 1
            else:
                raise ValueError("Illegal character: {}".format(self.string[i]))
        return len(tokens)

    def is_identifier(self, token):
        return 111, token

    def get_next_token(self):
        char = self.string[self.pointer]
        token=""
        if char.isdigit():
            token = char
            self.pointer += 1
            while self.pointer < len(self.string) and self.string[self.pointer].isdigit():
                token += self.string[self.pointer]
                self.pointer += 1
            return (100, token)
        elif char.isalnum() or char == '_':
            token = char
            self.pointer += 1
            while self.pointer < len(self.string) and (self.string[self.pointer].isalnum()
or self.string[self.pointer] == '_'):
                token += self.string[self.pointer]
                self.pointer += 1
            if token in self.keywords:
                return self.keywords[token]
            else:
                return self.is_identifier(token)
        elif char in self.operators or (self.string[self.pointer:self.pointer + 2]) in
self.operators:
            if (self.string[self.pointer:self.pointer + 2]) in self.operators:
                self.pointer += 2
                return self.operators[self.string[self.pointer-2:self.pointer]]
            else:
                self.pointer += 1
                return self.operators[char]
```

```
        elif char in self.delimiters:
            self.pointer += 1
            return self.delimiters[char]
        elif char.isspace():
            self.pointer += 1
            return self.get_next_token()

    def analyze(self):
        if self.pointer >= len(self.string):
            return None
        return self.get_next_token()

    def analyze_file(self, file_path):
        try:
            with open(file_path, 'r') as file:
                file_content = file.read()
                self.string = file_content
                self.token_num = self.count_tokens()
        except FileNotFoundError:
            print("File not found:", file_path)
class SymbolTableEntry:
    def __init__(self, name, ndim=0, place=None, array=None, offset=None):
        self.name = name
        self.ndim = ndim
        self.place = place
        self.array = array
        self.offset = offset
        self.in_place = []

    def set_dimensions(self, ndim):
        self.ndim = ndim

    def add_in_place(self, value):
        self.in_place.append(value)

    def __repr__(self):
        return f"SymbolTableEntry(name={self.name}, ndim={self.ndim},
place={self.place}, array={self.array}, offset={self.offset}, in_place={self.in_place})"

class SymbolTable:
    def __init__(self):
        self.table = {}
```

```
    def add_entry(self, name, ndim=0, place=None, array=None, offset=None):
        entry = SymbolTableEntry(name, ndim, place, array, offset)
        self.table[name] = entry
        return entry

    def get_entry(self, name):
        return self.table.get(name, None)

    def __repr__(self):
        return "\n".join(str(entry) for entry in self.table.values())
class SymbolTableEntry:
    def __init__(self, name, ndim=0, place=None, array=None, offset=None):
        self.name = name
        self.ndim = ndim
        self.place = place
        self.array = array
        self.offset = offset
        self.in_place = []

    def set_dimensions(self, ndim):
        self.ndim = ndim

    def add_in_place(self, value):
        self.in_place.append(value)

    def __repr__(self):
        return f"SymbolTableEntry(name={self.name}, ndim={self.ndim},
place={self.place}, array={self.array}, offset={self.offset}, in_place={self.in_place})"


def makelist(nextquad):
    return [nextquad]


class SymbolTable:
    def __init__(self):
        self.table = {}

    def add_entry(self, name, ndim=0, place=None, array=None, offset=None):
        entry = SymbolTableEntry(name, ndim, place, array, offset)
        self.table[name] = entry
```

```
        return entry

    def get_entry(self, name):
        return self.table.get(name, None)

    def __repr__(self):
        return "\n".join(str(entry) for entry in self.table.values())


class Parser:
    def __init__(self, lexer):
        self.lexer = lexer
        self.current_token = None
        self.step = 1
        self.temp_count = 0
        self.quadruples = []
        self.symbol_table = SymbolTable()
        self.quad_index = 1
        self.next_quad = 0
    def nextquad(self):
        return self.next_quad

    def makelist(self, index=None):
        if index is not None:
            return [index]
        else:
            return []

    def backpatch(self, list_, target):
        if list_ is None or list_ == []:
            self.quadruples[0][-1] = target
        else:
            for index in list_:
                self.quadruples[index][-1] = target

    def merge(self, list1, list2):
        if not list2:
            return list1
        elif not list1:
            return list2
        else:
            combined_list = list2 + list1
```

```
        return combined_list

    def eat(self, token_type):
        if self.current_token is not None and self.current_token[0] == token_type:
            self.current_token = self.lexer.analyze()

    def print_step(self, production):
        print(f"({self.step}) {production}")
        self.step += 1

    def emit(self, op, arg1, arg2, result):
        self.next_quad += 1
        self.quadruples.append([op, arg1, arg2, result])

    def newtemp(self):
        self.temp_count += 1
        return f"t{self.temp_count}"

    def factor(self):
        if self.current_token is None:
            return None
        token_type = self.current_token[0]
        if token_type == 100:
            self.print_step("factor ⟶ num")
            token_value = self.current_token[1]
            self.eat(100)
            return token_value
        elif token_type == 111:
            self.print_step("factor ⟶ loc")
            loc_place, loc_offset = self.loc()
            if loc_offset is None:
                return loc_place
            else:
                factor_place = self.newtemp()
                self.emit('=[]', f"{loc_place}[{loc_offset}]", '-', factor_place)
                return factor_place
        elif token_type == 81:
            self.print_step("factor ⟶ (expr)")
            self.eat(81)
            result = self.expression()
            self.eat(82)
            return result
```

```
        else:
            raise SyntaxError(f"Unexpected token: {self.current_token}")

    def loc(self):
        if self.current_token is None:
            return None, None
        self.print_step("loc ⟶ id resta")
        id_value = self.current_token[1]
        self.eat(111)
        entry = self.symbol_table.get_entry(id_value)
        if entry is None:
            entry = self.symbol_table.add_entry(id_value)
        inArray = id_value
        place, offset = self.resta(inArray)
        entry.place = place
        entry.offset = offset
        return place, offset

    def resta(self, inArray):
        if self.current_token is None:
            self.print_step("resta ⟶ ε")
            return inArray, None
        if self.current_token[0] == 88:
            self.print_step("resta ⟶ [elist]")
            self.eat(88)
            array = self.symbol_table.get_entry(inArray)
            array.place = inArray
            self.elist(array)
            self.eat(89)
            place = self.newtemp()
            self.emit('-', array.place, 'C', place)
            offset = self.newtemp()
            self.emit('*', array.offset, 'w', offset)
            return place, offset
        else:
            self.print_step("resta ⟶ ε")
            return inArray, None

    def elist(self, array):
        if self.current_token is None:
            return None
        self.print_step("elist ⟶ expr rest1")
```

```
        expr_place = self.expression()
        array.add_in_place(expr_place)
        rest1_inPlace = expr_place
        rest1_inNdim = 1
        array.offset = self.rest1(array, rest1_inNdim, rest1_inPlace)
        return None

    def rest1(self, array, inNdim, inPlace):
        if self.current_token is None:
            self.print_step("rest1 ⟶ ε")
            return inPlace
        if self.current_token[0] == 83:
            self.print_step("rest1 ⟶ , expr rest11")
            self.eat(83)
            expr_place = self.expression()
            t = self.newtemp()
            m = inNdim + 1
            self.emit('*', inPlace, f"n{m}", t)
            self.emit('+', t, expr_place, t)
            rest11_inPlace = t
            rest11_inNdim = m
            return self.rest1(array, rest11_inNdim, rest11_inPlace)
        else:
            self.print_step("rest1 ⟶ ε")
            return inPlace

    def term(self):
        if self.current_token is None:
            return None
        self.print_step("term ⟶ factor")
        left = self.factor()
        while self.current_token is not None and self.current_token[0] in (43, 44):
            self.eat(82)
            op = self.current_token[0]
            self.eat(op)
            right = self.factor()
            temp = self.newtemp()
            self.emit(self.lexer.get_operator_symbol(op), left, right, temp)
            left = temp
        return left

    def expression(self):
```

```
    if self.current_token is None:
        return None
    self.print_step("expr ⟶ term")
    left = self.term()
    while self.current_token is not None and self.current_token[0] in (41, 42):
        op = self.current_token[0]
        self.eat(op)
        right = self.term()
        temp = self.newtemp()
        self.emit(self.lexer.get_operator_symbol(op), left, right, temp)
        left = temp
    return left

def bool_expr(self):
    if self.current_token is None:
        return None
    self.print_step("bool ⟶ equality")
    return self.equality()

def equality(self):
    if self.current_token is None:
        return None, None
    self.print_step("equality ⟶ rel rest4")
    rel_truelist, rel_falselist = self.rel()
    return self.rest4(rel_truelist, rel_falselist)
def rel(self):
    if self.current_token is None:
        return None, None
    self.print_step("rel ⟶ expr rop_expr")
    expr_place = self.expression()
    rop_expr_inPlace = expr_place
    rop_expr_truelist, rop_expr_falselist = self.rop_expr(rop_expr_inPlace)
    rel_truelist = rop_expr_truelist
    rel_falselist = rop_expr_falselist
    return rel_truelist, rel_falselist

def rop_expr(self, inPlace):
    if self.current_token is None:
        return None, None
    if self.current_token[0] in (47, 48, 49, 50):
        self.print_step("rop_expr ⟶ rel_op expr")
        rel_op = self.current_token[0]
```

```
            self.eat(rel_op)
            expr_place = self.expression()
            truelist = self.makelist(self.nextquad())
            falselist = self.makelist(self.nextquad() + 1)
            self.emit(f'j{self.lexer.get_operator_symbol(rel_op)}',  inPlace,  expr_place,
'-')
            self.emit('j', '-', '-', '-')
            return truelist, falselist
        elif self.current_token[0] == 82:
            self.print_step("rop_expr ⟶ ε")
            nextquad = self.nextquad()
            truelist = self.makelist(nextquad)
            nextquad_plus_one = self.nextquad()+1
            falselist = self.makelist(nextquad_plus_one)
            self.emit('jnz', inPlace, '-', '-')
            self.emit('j', '-', '-', '-')
            return truelist, falselist
        else:
            raise SyntaxError(f"Unexpected token: {self.current_token}")

    def rest4(self, inTruelist, inFalselist):
        if self.current_token is None:
            return None, None
        if self.current_token[0] in (51, 52):
            self.print_step("rest4 ⟶ == rel rest41 | != rel rest41")
            self.rel()
            return None, None
        elif self.current_token[0] == 82:
            self.print_step("rest4 ⟶ ε")
            return inTruelist, inFalselist
        else:
            raise SyntaxError(f"Unexpected token: {self.current_token}")

    def stmts(self):
        if self.current_token is None:
            return self.makelist()
        self.print_step("stmts ⟶ stmt rest0")
        stmt_nextlist = self.stmt()
        self.rest0(stmt_nextlist)

    def stmt(self):
        if self.current_token is None:
```

```
        return self.makelist()
    if self.current_token[0] == 111:
        self.print_step("stmt ⟶ loc = expr;")
        loc_place, loc_offset = self.loc()
        self.eat(46)
        expr_place = self.expression()
        if loc_offset is None:
            self.emit('=', expr_place, '-', loc_place)
        else:
            self.emit('[]=', expr_place, '-', f"{loc_place}[{loc_offset}]")
        self.eat(84)
        return self.makelist()
    elif self.current_token[0] == 17:
        self.print_step("stmt ⟶ if (bool) stmt else stmt")
        self.eat(17)
        self.eat(81)
        bool_truelist, bool_falselist = self.bool_expr()
        self.eat(82)
        m1 = self.nextquad()
        stmt1_nextlist = self.stmt()
        n_nextlist = self.makelist(self.nextquad())
        self.emit('j', '-', '-', 0)
        self.eat(15)
        m2 = self.nextquad()
        stmt2_nextlist = self.stmt()
        self.backpatch(bool_truelist, m1)
        self.backpatch(bool_falselist, m2)
        return self.merge(stmt1_nextlist,self.merge(n_nextlist,stmt2_nextlist) )
    elif self.current_token[0] == 20:
        self.print_step("stmt ⟶ while (bool) stmt")
        self.eat(20)
        self.eat(81)
        m1 = self.nextquad()
        bool_truelist, bool_falselist = self.bool_expr()
        self.eat(82)
        m2 = self.nextquad()
        stmt1_nextlist = self.stmt()
        self.backpatch(stmt1_nextlist, m1)
        self.backpatch(bool_truelist, m2)
        self.emit('j', '-', '-', m1)
        return bool_falselist
    else:
```

```
            raise SyntaxError(f"Unexpected token: {self.current_token}")

    def rest0(self, inNextlist=None):
        if self.current_token is None:
            self.print_step("rest0 ⟶ ε")
            return inNextlist

        if self.current_token[0] in (111, 17, 20):
            self.print_step("rest0 ⟶ stmt rest0")
            m_quad = self.nextquad()
            self.stmt()
            self.backpatch(inNextlist, m_quad)
            stmt_nextlist = self.stmt()
            rest01_nextlist = self.rest0(stmt_nextlist)
            return rest01_nextlist
        else:
            self.print_step("rest0 ⟶ ε")
            return inNextlist


    def parse(self):
        self.current_token = self.lexer.analyze()
        self.stmts()
        return self.quadruples

lexer = Lexer()
lexer.analyze_file("source_code10.txt")
parser = Parser(lexer)
quadruples = parser.parse()

for idx, quad in enumerate(quadruples):
    print(f"{idx}: {quad}")
```