Лабораторная работа №7

```
from node import create_function, create_empty, create_var
from scanner import load_file, read_lex, next_lex, get_col, get_row
from tree import Tree

DEBUG_LEXER = False
DEBUG_TREE = True

tree: Tree
current_tree: Tree


def is_type(name: str):
    return name in ['SHORT', 'INT', 'LONG']


def program():
    global tree, current_tree
    tree = None
    current_tree = None
    if DEBUG_LEXER:
        print('program')
    tree = Tree()
    current_tree = tree
    while True:
        if read_lex().name == 'EOF':
            return
        if read_lex().name == 'VOID':
            function()
        elif is_type(read_lex().name):
            variable()
        else:
            err('Ожидался тип (short, int, long) или void')


def function():
    global tree, current_tree
```

```
    if DEBUG_LEXER:
        print('function')
    if next_lex().name != 'VOID':
        err('Ожидался void')
    lex = next_lex()
    if lex.name != 'ID':
        err(f'Ожидался идентификатор')
    name = lex.value
    new_tree = create_function(name)
    current_tree = current_tree.add_left(new_tree)
    if next_lex().name != 'ROUND_LEFT':
        err(f'Ожидался (')
    if next_lex().name != 'ROUND_RIGHT':
        err(f'Ожидался )')
    composite_operator()


def composite_operator():
    global tree, current_tree
    if DEBUG_LEXER:
        print('composite_operator')
    if next_lex().name != 'CURLY_LEFT':
        err('Ожидался {')
    if current_tree.right is not None:
        new_tree = create_empty()
        current_tree = current_tree.add_left(new_tree)
    new_tree = create_empty()
    current_tree = current_tree.add_right(new_tree)
    run = True
    while run is True and read_lex().name != 'EOF' and read_lex().name
!= 'CURLY_RIGHT':
        run = False
        if read_lex().name == 'ID':
            call_function()
            run = True
        if is_type(read_lex().name):
            variable()
```

```
            run = True
        if read_lex().name == 'IF':
            call_if()
            run = True
        if run is False:
            expression()
            run = True
    t = current_tree
    while t is not None and t.left is not None:
        t = t.up
        break
    current_tree = t.up
    if next_lex().name != 'CURLY_RIGHT':
        err('Ожидался }')


def call_function():
    global tree, current_tree
    if DEBUG_LEXER:
        print('call_function')
    lex = next_lex()
    if lex.name != 'ID':
        err('Ожидался идентификатор')
    if lex.value != 'print':
        err_sem(f'Функция {lex.value} не найдена')
    if next_lex().name != 'ROUND_LEFT':
        err('Ожидался (')
    expression()
    while read_lex().name == 'COMMA':
        next_lex()
        expression()
    if next_lex().name != 'ROUND_RIGHT':
        err('Ожидался )')
    if next_lex().name != 'SEMICOLON':
        err('Ожидался ;')
```

```
def expression():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression')
    expression_1()


def expression_1():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_1')
    expression_2()
    while read_lex().name == 'EQ' or read_lex().name == 'NOT_EQ':
        next_lex()
        expression_2()


def expression_2():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_2')
    expression_3()
    while read_lex().name == 'LESS' or read_lex().name == 'GREATER'
or read_lex().name == 'LESS_EQ' or read_lex().name ==
'GREATER_EQ':
        next_lex()
        expression_3()


def expression_3():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_3')
    expression_4()
    while read_lex().name == 'R_SHIFT' or read_lex().name ==
'L_SHIFT':
        next_lex()
```

```
        expression_4()


def expression_4():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_4')
    expression_5()
    while read_lex().name == 'PLUS' or read_lex().name == 'MINUS':
        next_lex()
        expression_5()


def expression_5():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_5')
    expression_6()
    while read_lex().name == 'STAR' or read_lex().name == 'SLASH' or
read_lex().name == 'PERCENT':
        next_lex()
        expression_6()


def expression_6():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_6')
    while read_lex().name == 'MINUS' or read_lex().name == 'PLUS':
        next_lex()
    expression_7()


def expression_7():
    global tree, current_tree
    if DEBUG_LEXER:
        print('expression_7')
```

```
    if read_lex().name == 'ROUND_LEFT':
        next_lex()
        expression()
        if next_lex().name != 'ROUND_RIGHT':
            err('Ожидался )')
    else:
        if read_lex().name == 'CURLY_LEFT':
            composite_operator()
        else:
            next_lex()


def variable():
    global tree, current_tree
    if DEBUG_LEXER:
        print('variable')
    lex = next_lex()
    if not is_type(lex.name):
        err('Ожидался тип (short, int, long)')
    type_var = lex.value
    f = True
    while f:
        lex = next_lex()
        if lex.name != 'ID':
            err('Ожидался идентификатор')
        var = create_var(lex.value, type_var)
        var.value = '0'
        fv = current_tree.find_var(var.name)
        if fv is not None and fv.node is not None and fv.node.type_object ==
'VAR' and fv.node.name == var.name:
            err_sem(f'Переменная {var.name} уже существует')
        current_tree = current_tree.add_left(var)
        if next_lex().name != 'ASSIGN':
            err('Ожидался =')
        expression()
        f = False
        if read_lex().name == 'COMMA':
```

```
            f = True
            next_lex()
    if next_lex().name != 'SEMICOLON':
        err('Ожидался ;')


def call_if():
    global tree, current_tree
    if DEBUG_LEXER:
        print('call_if')
    if next_lex().name != 'IF':
        err('Ожидался if')
    if next_lex().name != 'ROUND_LEFT':
        err('Ожидался (')
    expression()
    if next_lex().name != 'ROUND_RIGHT':
        err('Ожидался )')
    composite_operator()
    if read_lex().name == 'ELSE':
        next_lex()
        composite_operator()


def err(text: str):
    print(text, 'Найден:', read_lex(), 'Строка:', get_row(), 'Символ:',
get_col())
    exit(1)


def err_sem(text: str):
    print(text, 'Строка:', get_row(), 'Символ:', get_col())
    exit(1)


if __name__ == '__main__':
    load_file('examples/empty.c')
    program()
```

```
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
    print()

    load_file('examples/hex.c')
    program()
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
    if DEBUG_LEXER:
        print()
    print()

    load_file('examples/if.c')
    program()
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
    print()

    load_file('examples/math.c')
    program()
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
    print()

    load_file('examples/print.c')
    program()
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
    print()

    load_file('examples/types.c')
    program()
    if DEBUG_TREE and tree is not None:
        tree.get_root().print(0)
```

Примеры работы

```
void main() {
    short a = 1;
    int b = 2;
    long c = 3, d = 4;
    print(a + b + c + d);
}
```

```
examples/types.c
this=(None)
this=(FUNCTION main params=[])
    this=(EMPTY)
    this=(VAR a value=0)
    this=(VAR b value=0)
    this=(VAR c value=0)
    this=(VAR d value=0)
```