Örebro University

Compilers and Interpreters

Assignment 6

Author:

Linus Baumgärtner
lbaumgaertner@culba.de

Marko Reichhart
marko.reichhart@web.de

October 20, 2022

Executing syntax trees

The function `execute` as seen in listing 1 implements a recursive execution function of the syntax tree. According to the type of the tree node the specific operation is executed to its children. If the node is a leaf node the `leaf_value` is returned. The function is executed by the statement `!exe`.

Listing 1: Recursive code execution function

```
int execute(TreeNode* p) {
    if (p != 0) {
        switch (p->type) {
            case ' ':
                break;
            case NUM:
                return p->leaf_value;
            case ID:
                if (symtable[p->leaf_value].value_initialized){
                    return symtable[p->leaf_value].value;
                }
                else
                    printf("Error: variable %s not initialized!", symtable[p->leaf_value].lexeme);
                break;
            case '+':
                return execute(p->args[0]) + execute(p->args[1]);
            case '-':
                return execute(p->args[0]) - execute(p->args[1]);
            case '*':
                return execute(p->args[0]) * execute(p->args[1]);
            case '/':
                return execute(p->args[0]) / execute(p->args[1]);
            case '%':
                return execute(p->args[0]) % execute(p->args[1]);
            case '^':
                return pow(execute(p->args[0]), execute(p->args[1]));
            case '&':
                return execute(p->args[0]) & execute(p->args[1]);
            case '|':
                return execute(p->args[0]) | execute(p->args[1]);
            case '<':
                return execute(p->args[0]) < execute(p->args[1]);
            case '>':
                return execute(p->args[0]) > execute(p->args[1]);
            case '=':
            {
                symtable[p->args[0]->leaf_value].value = execute(p->args[1]);
                symtable[p->args[0]->leaf_value].value_initialized = true;
                break;
            }
            case '?':
                return execute(p->args[0]) ? execute(p->args[1]) : execute(p->args[2]);
            case IF:
            {
                int condition = execute(p->args[0]);
                if (condition)
                    execute(p->args[1]);
                else
                    execute(p->args[2]);
                break;
            }
            case WHILE:
                while (execute(p->args[0]))
                    execute(p->args[1]);
```

```
                break;
            case PRINT:
                printf("%d\n", execute(p->args[0]));
                break;
            case READ:
            {
                int n;
                std::cin >> n;
                symtable[p->args[0]->leaf_value].value = n;
                symtable[p->args[0]->leaf_value].value_initialized = true;
                break;
            }
            case ';':
                execute(p->args[0]);
                if (p->args[1] != 0)
                    execute(p->args[1]);
                break;
        }
    }
    return 0;
}
```