Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №10-2

по курсу: «Языки и методы программирования»

«Реализация итераторов на языке C++»

Выполнил:
Студент группы ИУ9-21Б

Проверил:
Посевин Д. П.

Москва, 2024

1. Цель

Данная работа предназначена для приобретения навыков разработки контейнерных классов с итераторам.

2. Персональный вариант

Арифметическая формула, представленная в виде дерева и собираемая из примитивных деревьев, представляющих целочисленные константы, с помощью перегруженных операций «+», «-», «*» и «/». У формулы должен быть двунаправленный итератор по константам (листьям дерева) с возможностью их изменения.

3. Решение

3.1. Код

```
#include <bits/stdc++.h>

#include <utility>

using namespace std;

template <typename T, typename E> class Stream {
  private:
    T stream_;
    size_t pointer_;

    pair<E, bool> get(size_t p) {
        if (!(p < stream_.size()))
            return {E(), false};

        return {stream_[p], true};
    }

  public:
    Stream() : stream_{}, pointer_(0) {}
    Stream(T s) : stream_(std::move(s)), pointer_(0) {}

    E peek() { return get(pointer_).first; }

    vector<E> peekn(size_t n) {
        vector<E> result;

        for (size_t i = 0; i < n; ++i) {
            auto [k, valid] = get(pointer_ + i);
            if (!valid)
```

```
                break;
            result.push_back(k);
        }

        return result;
    }

    E next() { return get(pointer_++).first; }

    void skip(size_t k = 1) { pointer_ += k; }

    bool empty() { return pointer_ >= stream_.size(); }
};

enum TokenType { ERROR, PLUS, MINUS, MUL, DIV, NUMBER, EOP };

struct Token {
    string value;
    TokenType type;
};

class Lexer {
  private:
    typedef vector<Token>::iterator iterator;

    bool error_;

    vector<Token> tokens_;
    Stream<string, char> stream_;

    vector<Token> specials_ = {{"+", PLUS}, {"-", MINUS}, {"*",
    ↪  MUL}, {"/", DIV}};

    Token scanNumber() {
        string result(1, stream_.next());

        for (; !stream_.empty(); stream_.skip()) {
            char cur = stream_.peek();

            if (!isdigit(cur))
                break;

            result.push_back(cur);
        }
```

```
        return {result, NUMBER};
    }

    Token scanSpecial() {
        int max_n = specials_[0].value.length();
        vector<char> symb = stream_.peekn(max_n);

        for (auto [value, type] : specials_) {
            if (value.length() <= symb.size()) {
                bool equal = true;

                for (size_t i = 0; i < value.length(); ++i) {
                    if (value[i] != symb[i]) {
                        equal = false;
                        break;
                    }
                }

                if (!equal)
                    continue;

                stream_.skip(value.size());
                return {value, type};
            }
        }

        error();

        return {"(T-T)", ERROR};
    }

    void error() { error_ = true; }

    void tokenize() {
        while (!stream_.empty() && !error_) {
            if (isdigit(stream_.peek()))
                tokens_.push_back(scanNumber());
            else if (isspace(stream_.peek()))
                stream_.next();
            else
                tokens_.push_back(scanSpecial());
        }
    }

public:
```

```
    Lexer(const string &s) : error_(false), stream_(s) {
        auto comp = [&](auto left, auto right) {
            auto [left_k, left_v] = left;
            auto [right_k, right_v] = right;
            return left_k.size() > right_k.size() ||
                   (left_k.size() == right_k.size() && left_v <
                    ↪  right_v);
        };

        sort(specials_.begin(), specials_.end(), comp);

        tokenize();
    }

    vector<Token> tokens() { return this->tokens_; }
    bool success() { return !error_; }

    iterator begin() { return this->tokens_.begin(); }
    iterator end() { return this->tokens_.end(); }
};

class ArithmeticFormula {
  private:
    vector<int> iter;
    Stream<vector<Token>, Token> stream;

  public:
    ArithmeticFormula(string s) {
        Lexer l(s);
        assert(l.success());

        stream = l.tokens();

        parseExpr();
    }

    void parseExpr() {
        parseTerm();

        while (stream.peek().type == DIV || stream.peek().type
         ↪  == MUL) {
            stream.skip();
            parseTerm();
        }
    }
```

```
    void parseTerm() {
        parseFactor();

        while (stream.peek().type == PLUS || stream.peek().type
        ↪  == MINUS) {
            stream.skip();
            parseFactor();
        }
    }

    void parseFactor() {
        if (stream.peek().type == NUMBER) {
            iter.push_back(stoi(stream.next().value));
            return;
        }

        assert(0);
    }

    auto begin() { return iter.begin(); }
    auto end() { return iter.end(); }
    auto rbegin() { return iter.rbegin(); }
    auto rend() { return iter.rend(); }
};

int main() {
    ArithmeticFormula f("13 / 9 * 7 + 9 -  12");

    for (auto it = f.begin(); it != f.end(); ++it) {
        cout << (*it) << ' ';
    }
    cout << '\n';

    int i = 0;
    for (auto it = f.rbegin(); it != f.rend(); ++it) {
        *it = i++;
    }

    for (auto it = f.end(); it != f.begin(); --it) {
        cout << (*(it - 1)) << ' ';
    }
    cout << '\n';
}
```

Код 1: main.cc

3.2. Скриншоты

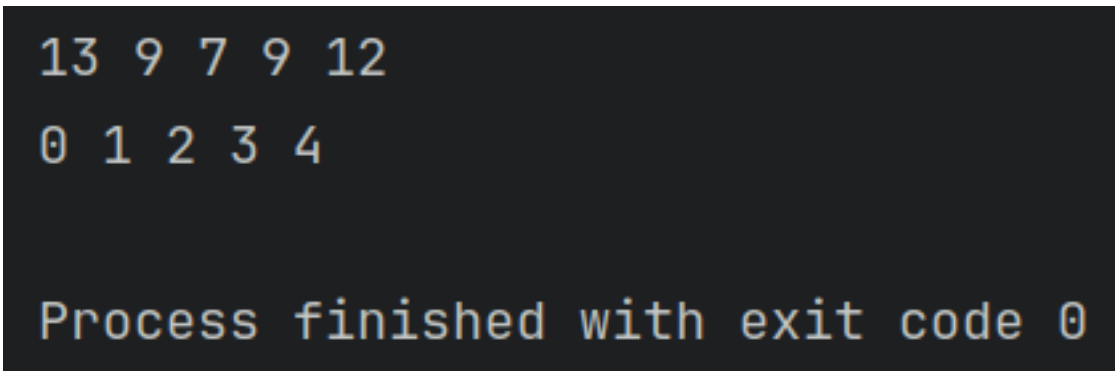

Рис. 1: Пример работы программы