**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: "Польские нотации. Стек и очередь"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0324 |  | Гордиенко Т.Е. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2021

**Цель работы**: получение практических навыков работы со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных.

**Постановка задачи**:

Необходимо написать программу, которая выполняет следующее:

1.   Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и \* с файла.

2.   Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и \* с файла.

3.   Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

\*4.   Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (прямой и обратной). Задание и ответы к ним необходимо вывести в отдельные файлы (ответы должны быть максимально подробными).

Выполнение задачи:

|  |
| --- |
| #include <iostream> |
|  | #include <stack> |
|  | #include <string> |
|  | #include <fstream> |
|  | using namespace std; |
|  |  |
|  | int getPriority(char C) |
|  | { |
|  | if (C == '+') { |
|  | return 1; |
|  | } |
|  | else if (C == '-') { |
|  | return 2; |
|  | } |
|  | else if (C == '\*' || C == '/') |
|  | return 3; |
|  | return 0; |
|  | } |
|  |  |
|  | char generateOperator() { |
|  | int num = rand() % 3; |
|  | switch(num) { |
|  | case 0: |
|  | return '+'; |
|  | case 1: |
|  | return '-'; |
|  | case 2: |
|  | return '\*'; |
|  | } |
|  | } |
|  |  |
|  | string infixToPostfix(string in\_line) |
|  | { |
|  | in\_line = '(' + in\_line + ')'; |
|  | int l = in\_line.size(); |
|  | stack<char> operators; |
|  | string output; |
|  | for (int i = 0; i < l; i++) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9')//если цифра |
|  | output += in\_line[i]; |
|  | else if (in\_line[i] == '(') |
|  | operators.push('('); |
|  | else if (in\_line[i] == ')') { |
|  | while (operators.top() != '(') { |
|  | output += operators.top(); |
|  | operators.pop(); |
|  | } |
|  | operators.pop(); |
|  | } |
|  | else//если оператор |
|  | { |
|  | while (getPriority(in\_line[i]) <= getPriority(operators.top())) |
|  | { |
|  | output += operators.top(); |
|  | operators.pop(); |
|  | } |
|  | operators.push(in\_line[i]); |
|  | } |
|  | } |
|  | return output; |
|  | } |
|  |  |
|  | string infixToPostfixFile(string in\_line, ofstream \*file) { |
|  | in\_line = '(' + in\_line + ')'; |
|  | int l = in\_line.size(); |
|  | stack<char> operators; |
|  | string output; |
|  | for (int i = 0; i < l; i++) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9') {//если цифра |
|  | output += in\_line[i]; |
|  | \*file << output << '\n'; |
|  | } |
|  | else if (in\_line[i] == '(') { |
|  | operators.push('('); |
|  | } |
|  | else if (in\_line[i] == ')') { |
|  | while (operators.top() != '(') { |
|  | output += operators.top(); |
|  | \*file << output << '\n'; |
|  | operators.pop(); |
|  | } |
|  | operators.pop(); |
|  | } |
|  | else//если оператор |
|  | { |
|  | while (getPriority(in\_line[i]) <= getPriority(operators.top())) |
|  | { |
|  | output += operators.top(); |
|  | \*file << output << '\n'; |
|  | operators.pop(); |
|  | } |
|  | operators.push(in\_line[i]); |
|  | } |
|  | } |
|  | return output; |
|  | } |
|  |  |
|  | string infixToPrefix(string infix) |
|  | { |
|  | int l = infix.size(); |
|  | reverse(infix.begin(), infix.end()); |
|  | for (int i = 0; i < l; i++) { |
|  | if (infix[i] == '(') { |
|  | infix[i] = ')'; |
|  | i++; |
|  | } |
|  | else if (infix[i] == ')') { |
|  | infix[i] = '('; |
|  | i++; |
|  | } |
|  | } |
|  | string prefix = infixToPostfix(infix); |
|  | reverse(prefix.begin(), prefix.end()); |
|  | return prefix; |
|  | } |
|  |  |
|  | string infixToPrefixFile(string infix, ofstream \*file) { |
|  | int l = infix.size(); |
|  | reverse(infix.begin(), infix.end()); |
|  | for (int i = 0; i < l; i++) { |
|  | if (infix[i] == '(') { |
|  | infix[i] = ')'; |
|  | i++; |
|  | } |
|  | else if (infix[i] == ')') { |
|  | infix[i] = '('; |
|  | i++; |
|  | } |
|  | } |
|  | string prefix = infixToPostfix(infix); |
|  | reverse(prefix.begin(), prefix.end()); |
|  | \*file << prefix << '\n'; |
|  | return prefix; |
|  | } |
|  |  |
|  | int calculatePostfix(string in\_line) { |
|  | stack<int> counters; |
|  | for (int i = 0; i < in\_line.length(); i++) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9') { |
|  | counters.push(in\_line[i] - '0'); |
|  | } |
|  | else { |
|  | int op2 = counters.top(); |
|  | counters.pop(); |
|  | int op1 = counters.top(); |
|  | counters.pop(); |
|  | switch (in\_line[i]){ |
|  | case '+': |
|  | counters.push(op1 + op2); |
|  | break; |
|  | case '-': |
|  | counters.push(op1 - op2); |
|  | break; |
|  | case '\*': |
|  | counters.push(op1 \* op2); |
|  | break; |
|  | case '/': |
|  | counters.push(op1 / op2); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | } |
|  | return counters.top(); |
|  | } |
|  |  |
|  | int calculatePostfixFile(string in\_line, ofstream \*file) { |
|  | stack<int> counters; |
|  | for (int i = 0; i < in\_line.length(); i++) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9') { |
|  | counters.push(in\_line[i] - '0'); |
|  | } |
|  | else { |
|  | int op2 = counters.top(); |
|  | counters.pop(); |
|  | int op1 = counters.top(); |
|  | counters.pop(); |
|  | switch (in\_line[i]) { |
|  | case '+': |
|  | counters.push(op1 + op2); |
|  | \*file << to\_string(op1) << " + " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '-': |
|  | counters.push(op1 - op2); |
|  | \*file << to\_string(op1) << " - " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '\*': |
|  | counters.push(op1 \* op2); |
|  | \*file << to\_string(op1) << " \* " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '/': |
|  | counters.push(op1 / op2); |
|  | \*file << to\_string(op1) << " / " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | } |
|  | \*file << '\n'; |
|  | return counters.top(); |
|  | } |
|  |  |
|  | int calculatePrefix(string in\_line){ |
|  | stack<int> counters; |
|  | for (int i = in\_line.size() - 1; i >= 0; i--) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9') |
|  | counters.push(in\_line[i] - '0'); |
|  | else { |
|  | int op1 = counters.top(); |
|  | counters.pop(); |
|  | int op2 = counters.top(); |
|  | counters.pop(); |
|  | switch (in\_line[i]) { |
|  | case '+': |
|  | counters.push(op1 + op2); |
|  | break; |
|  | case '-': |
|  | counters.push(op1 - op2); |
|  | break; |
|  | case '\*': |
|  | counters.push(op1 \* op2); |
|  | break; |
|  | case '/': |
|  | counters.push(op1 / op2); |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | return counters.top(); |
|  | } |
|  |  |
|  | int calculatePrefixFile(string in\_line, ofstream \*file) { |
|  | stack<int> counters; |
|  | for (int i = in\_line.size() - 1; i >= 0; i--) { |
|  | if (in\_line[i] >= '0' && in\_line[i] <= '9') |
|  | counters.push(in\_line[i] - '0'); |
|  | else { |
|  | int op1 = counters.top(); |
|  | counters.pop(); |
|  | int op2 = counters.top(); |
|  | counters.pop(); |
|  | switch (in\_line[i]) { |
|  | case '+': |
|  | counters.push(op1 + op2); |
|  | \*file << to\_string(op1) << " + " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '-': |
|  | counters.push(op1 - op2); |
|  | \*file << to\_string(op1) << " - " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '\*': |
|  | counters.push(op1 \* op2); |
|  | \*file << to\_string(op1) << " \* " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | case '/': |
|  | counters.push(op1 / op2); |
|  | \*file << to\_string(op1) << " / " << to\_string(op2) << " = " << to\_string(counters.top()) << '\n'; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | \*file << '\n'; |
|  | return counters.top(); |
|  | } |
|  |  |
|  | int calculateInfix(string in\_line) { |
|  | in\_line = infixToPostfix(in\_line); |
|  | return calculatePostfix(in\_line); |
|  | } |
|  |  |
|  | void generateTest() { |
|  | srand(time(NULL)); |
|  | cout << '\n' << "Место создания файлов(d:\\folder\\folder\\): "; |
|  | string path; |
|  | getline(cin, path); |
|  | ofstream ask(path + "ask.txt"), key(path + "key.txt"), ans(path + "ans.txt"); |
|  | if (!ask.is\_open() || !ans.is\_open() || !key.is\_open()) { |
|  | std::cout << "Ошибка создания файла" << '\n'; |
|  | return; |
|  | } |
|  | int count, temp; |
|  | cout << "Количество вариантов: "; |
|  | cin >> count; |
|  | for (int i = 0; i != count; i++) { |
|  | path = ""; |
|  | ask << "Вариант " << i + 1 << '\n'; |
|  | ans << "Вариант " << i + 1 << '\n'; |
|  | key << "Вариант " << i + 1 << '\n'; |
|  | path += to\_string(rand() % 9 + 1); |
|  | for (int i = 0; i != 5; i++) { |
|  | path += generateOperator(); |
|  | path += to\_string(rand() % 9 + 1); |
|  | } |
|  | cout << path << '\n'; |
|  | ask << path << '\n'; |
|  | ask << "1) Преобразовать выражение в постфиксную и префиксную запись" << '\n'; |
|  | ask << "2) Решить это выражение" << '\n'; |
|  | key << infixToPostfixFile(path, &ans) << '\n' << '\n'; |
|  | key << infixToPrefixFile(path, &ans) << '\n' << '\n'; |
|  | key << calculatePostfixFile(infixToPostfix(path), &ans) << '\n' << '\n'; |
|  | key << calculatePrefixFile(infixToPrefix(path), &ans) << '\n' << '\n'; |
|  | } |
|  | ask.close(); |
|  | ans.close(); |
|  | key.close(); |
|  | } |
|  |  |
|  | int main() { |
|  | setlocale(NULL, ""); |
|  | srand(time(NULL)); |
|  | std::string infix, postfix, prefix; |
|  | std::cout << "Введите инфиксное выражение (допускаются числа от 0 до 9 и операторы /\*+-, ввод без пробелов):"; |
|  | std::getline(std::cin, infix); |
|  | postfix = infixToPostfix(infix); |
|  | prefix = infixToPrefix(infix); |
|  | std::cout << "инфиксное выражение: " << infix << '\n'; |
|  | std::cout << "постфиксное выражение: " << postfix << '\n'; |
|  | std::cout << "префиксное выражение: " << prefix << '\n' << '\n'; |
|  | std::cout << "результат для инфиксного: " << calculateInfix(infix) << '\n'; |
|  | std::cout << "результат для постфиксного: " << calculatePostfix(postfix) << '\n'; |
|  | std::cout << "результат для префиксного: " << calculatePostfix(postfix) << '\n'; |
|  | generateTest(); |
|  | return 0; |
|  | } |