Odwrotna Notacja Polska

Sprawozdanie – Łukasz Konieczny | LK4 | Lab 3

1. Wstęp teoretyczny

Odwrotna notacja polska została wymyślona przez Charlesa Hamblina, jako „odwrócenie" notacji polskiej, czyli bez nawiasowego sposobu zapisu działań matematycznych, autorstwa Jana Łukasiewicza.

Pozwala ona na zapisanie dowolnego działania matematycznego bez używania nawiasów, przy jednoczesnym zachowaniu odpowiedniej kolejności wykonywania działań.

Jest to tak zwana „notacja post-fixowa", co oznacza, że operatory matematyczne umieszczone są na końcu działania, po operandach.

W przeciwieństwie do tradycyjnej, „in-fixowej" notacji którą używamy na co dzień, jest ona dużo prostsza do zinterpretowania i wykonania przez komputer, dlatego znalazła szerokie zastosowanie w informatyce.

Wykorzystuje ona „stos" – strukturę danych typu LIFO (last-in-first-out) czyli taką, w której układamy elementy na sobie i mamy dostęp tylko do leżącego na jego szczycie.
Aby dostać się do któregoś z wcześniej dodanych elementów, musimy najpierw zdjąć ze stosu te dodane najwcześniej, ponieważ znajdują się one na jego szczycie i blokują nam dostęp do innych.

Aby zamienić działanie w notacji in-fixowej na ONP, wykorzystujemy następujący algorytm:

Wyrażenie arytmetyczne odczytujemy od strony lewej do prawej.

- Jeśli dojdziemy do końca wyrażenia, to ze stosu operatorów pobieramy operatory i przenosimy je kolejno na wyjście aż do wyczyszczenia stosu. Algorytm kończymy.
- Jeśli odczytanym elementem jest symbol zmiennej, to przenosimy go na wyjście.
- Jeśli odczytanym elementem jest nawias otwierający, to umieszczamy go na stosie.
- Jeśli odczytanym elementem jest nawias zamykający, to ze stosu przesyłamy na wyjście wszystkie operatory, aż do napotkania nawiasu otwierającego, który usuwamy ze stosu.
- Jeśli odczytanym elementem jest operator, to:

  dopóki na stosie jest jakiś operator i:

  odczytany operator ma łączność lewostronną oraz priorytet niższy lub równy operatorowi na stosie

  lub odczytany operator ma łączność prawostronną i priorytet niższy od operatora na stosie,

  to pobieramy ze stosu operator i przesyłamy go na wyjście

- Po tej operacji odczytany operator umieszczamy na stosie.
- Kontynuujemy od początku z następnym elementem.

// Algorytm dostępny na stronie https://eduinf.waw.pl/inf/alg/001_search/0102.php

Wykonanie działania zapisanego w ONP jest już zdecydowanie prostsze.

Działanie odczytujemy od lewej do prawej.

- Jeśli odczytaną wartością jest liczba, umieszczamy ją na stosie.
- Jeśli natomiast operator, pobieramy ze stosu oczekiwaną przez niego liczbę elementów, wykonujemy działanie a następnie jego wynik umieszczamy na stosie.
- Gdy dojdziemy do końca poprawnie zapisanego działania, na stosie powinna znajdować się już tylko jedna liczba, która stanowi wynik.

Jak więc widzimy, ONP pozwala nam w prosty sposób zaprogramować kalkulator „naukowy” (respektujący kolejność wykonywania działań), dlatego jest tak popularna wśród informatyków.

Powyższe algorytmy cechuje złożoność O(n), gdzie n to liczba elementów w działaniu.

2. Implementacja

Oba z powyższych algorytmów zaimplementowałem w języku C++.

Skorzystałem z następujących bibliotek:

```cpp
#include <iostream>
#include <stack>
#include <string>
#include <vector>
#include <cmath>
#include <fstream>
```

Oprócz bibliotek wejścia/wyjścia korzystam z biblioteki string (aby ułatwić sobie przechowywanie działania w programie), stack (ponieważ potrzebujemy wykorzystać stos) a także vector (posłuży mi do przechowywania działania podzielonego na poszczególne elementy) i cmath (do obliczania wartości działania MODULO dla liczb typu double oraz do potęgowania).

Na początku zdefiniowałem kilka funkcji pomocniczych, aby uprościć dalszy kod:

```cpp
int getPriority(char c) {
    switch (c) {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
    case 'm':
        return 2;

    case 's':
    case '^':
        return 3;

    case '(':
    case ')':
        return 0;

    default:
        return -1;
    }
}
```

```cpp
void popOperator(stack<char>& operators, string& result) {
    if (char top = operators.top(); top == 's')
        result += "sqrt";
    else if (top == 'm')
        result += "mod";
    else
        result += top;

    operators.pop();
    result += ' ';
}
```

```cpp
void popOperands(stack<double>& operands, double& left, double& right) {
    right = operands.top();
    operands.pop();
    left = operands.top();
    operands.pop();
}
```

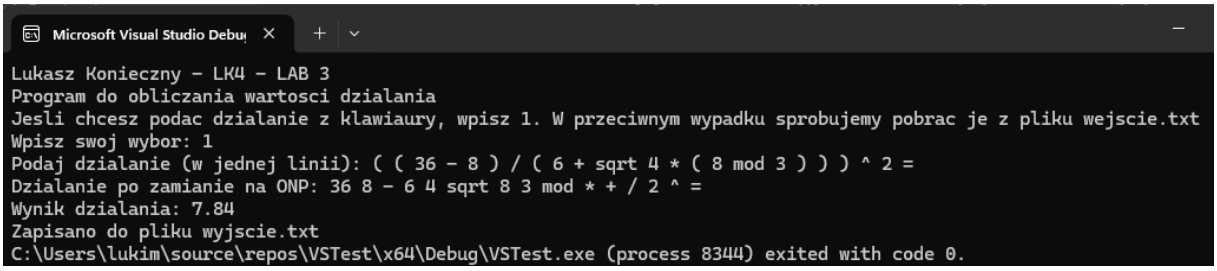
```
vector<string> splitString(string param) {
    vector<string> parsed;

    string current = "";
    for (int i = 0; i < param.length(); i++) {
        if (param[i] == ' ') {
            parsed.push_back(current);
            current = "";
        }
        else {
            current += param[i];
        }
    }
    parsed.push_back(current);
    return parsed;
}
```

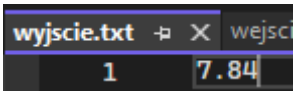
- Funkcja getPriority zwraca liczbowy odpowiednik priorytetu podanego operatora.
- popOperator ściąga ze stosu operator oraz dopisuje jego symbol do wynikowego stringa.
- popOperands ściąga ze stosu dwa operandy oraz umieszcza je w podanych zmiennych left i right, aby można było wykorzystać je w obliczeniach.
- splitString używam do podzielenia stringa zawierającego działanie na „tokeny”, dzieląc go po spacji. Wynik operacji umieszczany jest w obiekcie typu vector (mogłaby to być tablica, jednak to wymagałoby policzenia ilości tokenów i utworzenia tablicy o stosownym rozmiarze przez faktycznym dzieleniem).

Następnie zdefiniowałem funkcje createONP oraz calculateONP, które realizują zadania prezentowane przez wymienione wcześniej algorytmy, z pomocą funkcji pomocniczych o których mówiłem wcześniej.

Program obsługuje przekazanie przez użytkownika działania w konsoli, lub odczytanie go z pliku.

Zrzuty ekranu:





UWAGA! Program wykorzystuje składnię języka C++ w wersji 17, aby go uruchomić należy upewnić się, że wybrany przez nas kompilator obsługuję tą wersję języka lub nowszą!

```
if (int p = getPriority(first); p != -1)
```
// Zapis możliwy dopiero od C++ 17

3. Wnioski

Podczas zajęć zapoznaliśmy się z Odwrotną Notacją Polską – bardzo przydatnym w informatyce sposobem zapisu działań matematycznych.

Poznaliśmy algorytmy pozwalające zarówno obliczyć wartość wyrażenia ONP, jak i przekształcić wyrażenie in-fixowe na ONP.

Zaimplementowaliśmy je w języku C++, dzięki czemu stworzyliśmy kalkulator obsługujący kolejność wykonywania działań, oraz poznaliśmy biblioteki stack i vector.

Bibliografia:

- Materiał z wykładów o strukturach danych
- Instrukcje na platformie Delta
- Opis i przykład wykonania algorytmów ONP na: Algorytmy i Struktury Danych - Odwrotna Notacja Polska (eduinf.waw.pl)
- Kalkulator ONP do sprawdzenia poprawności działania programu: Kalkulator ONP (k144.github.io)
- Artykuł o ONP na Wikipedii