

Projekti 1 – Postfix-laskin  
Tietorakenteen ja algoritmit

Caritalin

PROJEKTI 1 – POSTFIX-LASKIN

Helmikuu 2024

Tietotekniikan tutkinto-ohjelma

Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu

Tietotekniikan tutkinto-ohjelma

Ohjelmistotekniikka

CARITALIN

Projekti 1 – Postfix-laskin

Tietorakenteet ja algoritmit

Projekti 30 sivua, joista 1 sivu tuntikirjanpitoa

Helmikuu 2024

Tässä harjoitustyössä tehdään ohjelmakoodi postfix-laskimen eri variaatioista tehtävänannon inkrementtien vaatimusten mukaisesti.

Harjoitustyö on tehty viidellä inkrementillä ja tämän myötä olen lähtenyt hakemaan projektityöstä arvosanaa 5.

SISÄLLYS

[1 1. INKREMENTTI 5](#_Toc158305046)

[1.1 Yhden pisteen työn vaatimukset 5](#_Toc158305047)

[1.1.1 Toteutus 5](#_Toc158305048)

[1.1.2 Toiminta 6](#_Toc158305049)

[1.1.3 1. Inkrementin lähdekoodi 7](#_Toc158305050)

[2 2. INKREMENTTI 9](#_Toc158305051)

[2.1 Kahden pisteen työn vaatimukset 9](#_Toc158305052)

[2.1.1 Toteutus 9](#_Toc158305053)

[2.1.2 Toiminta 10](#_Toc158305054)

[2.1.3 2. Inkrementin lähdekoodi 11](#_Toc158305055)

[3 3. INKREMENTTI 14](#_Toc158305056)

[3.1 Kolmen pisteen työn vaatimukset 14](#_Toc158305057)

[3.1.1 Toteutus 14](#_Toc158305058)

[3.1.2 Toiminta 15](#_Toc158305059)

[3.1.3 3. Inkrementin lähdekoodi 15](#_Toc158305060)

[4 4. INKREMENTTI 18](#_Toc158305061)

[4.1 Neljän pisteen työn vaatimukset 18](#_Toc158305062)

[4.1.1 Toteutus 18](#_Toc158305063)

[4.1.2 Toiminta 19](#_Toc158305064)

[4.1.3 4. Inkrementin lähdekoodi 19](#_Toc158305065)

[5 5. INKREMENTTI 23](#_Toc158305066)

[5.1 Viiden pisteen työn vaatimukset 23](#_Toc158305067)

[5.1.1 Toteutus 23](#_Toc158305068)

[5.1.2 Toiminta 24](#_Toc158305069)

[5.1.3 5. inkrementin lähdekoodi 25](#_Toc158305070)

[POHDINTA 29](#_Toc158305071)

[Tuntikirjanpito 30](#_Toc158305072)

JOHDANTO

Tässä projektissa tehdään postfix-laskimen eri variaatioita tehtävänannon inkrementtien mukaisesti. Tehtävässä kaikkien koodit ovat nimetty seuraavalla tavoin:

* postfix\_calculator1.cpp
* postfix\_calculator2.cpp
* postfix\_calculator3.cpp
* postfix\_calculator4.cpp
* postfix\_calculator5.cpp

Näissä postfix-calculator nimen perässä oleva numero kertoo minkä inkrementin koodi on kyseessä. Inkrementeissä jatketaan koodia aina edellisen inkrementin pohjalle, joten jokaisessa ohjelmakoodissa on edellisen koodin toiminnallisuudet. Viidennessä inkrementissä koodin ajaminen komentoriviltä tapahtuu vain eri tavalla, sillä siihen on määritelty lisäkomento -p (prompt), joka tuo 1-4 inkrementtien toiminnallisuudet lisäksi viidennen inkrementin laajempaa komentoskaalaa.

Koodit voi ajaa komentoriviltä seuraavalla komennolla tehtävänannon mukaisesti laittaen postfix\_calculator perään lisää vain inkrementin numeron:

g++ postfix\_calculator.cpp -o postfix\_calculator

(Esim. g++ postfix\_calculator1.cpp -o postfix\_calculator1)

Kaikki ohjelmakoodit löytyvät GitHubista osoitteesta:

<https://github.com/caritalin/projekti1_tietorakenteet_ja_algoritmit.git>

# 1. INKREMENTTI

## Yhden pisteen työn vaatimukset

Tehtävänantona on toteuttaa toimiva Postfix-laskin luennoilla läpikäytyjä ohjelmakoodeja soveltaen. Tähän versioon ei tehdä vielä mitään lisätoimintoja.

### Toteutus

Tietokoodi toteuttaa yksinkertaisen PostFix-laskimen C++-koodikieltä käyttäen. Koodin toiminta vaiheittain:

1. Alustetaan tarvittavat kirjastot: iostream, stack, sstream ja cmath. Näitä tarvitaan syötteiden lukemiseen, tietojen tallentamiseen pinoon, merkkijonojen käsittelyyn ja matemaattisten toimintojen suorittamiseen.
2. Määritellään isOperator-funktio tarkistamaan, onko annettu merkkijono operaattori (eli +, -, \*, /).
3. Määritellään isNumber-funktio tarkistamaan, onko annettu merkkijono numero.
4. Määritellään evaluateOperation-funktio, joka suorittaa annetun operaation kahdelle operandille ja palauttaa tuloksen.
5. Määritellään evaluatePostFix-funktio, joka ottaa PostFix-lausekkeen merkkijonona, arvioi sen ja palauttaa lopputuloksen.
6. Pääohjelmassa pyydetään käyttäjältä syötettä PostFix-lausekkeena.
7. Käyttäjän antama lauseke arvioidaan evaluatePostFix-funktiolla.
8. Lopputulos tulostetaan näytölle, jos se on kelvollinen. Jos arviointi epäonnistuu tai lauseke on virheellinen, tulostetaan virheilmoitus.

Kokonaisuudessaan ohjelma lukee PostFix-lausekkeen käyttäjältä, arvioi sen käyttäen pino-operaatioita ja tulostaa lopputuloksen.

### Toiminta

1. Yhteenlasku (+)

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

1. Vähennyslasku (-)

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

1. Jakolasku (/)

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

1. Kertolasku (\*)

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

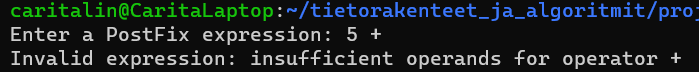
Virheilmoitukset:

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti



### 1. Inkrementin lähdekoodi

#include <iostream>

#include <stack>

#include <sstream>

#include <cmath>

using namespace std;

// Function to check if a token is an operator

bool isOperator(const string& token) {

    return token == "+" || token == "-" || token == "\*" || token == "/";

}

bool isNumber(const string& token) {

    stringstream ss(token);

    double num;

    // Try to extract a number from the stringstream

    return ss >> num && ss.eof(); // Return true if extraction successful and no further characters left

}

// Function to evaluate an operation

double evaluateOperation(double operand1, double operand2, const string& op) {

    if (op == "+")

        return operand1 + operand2;

    else if (op == "-")

        return operand1 - operand2;

    else if (op == "\*")

        return operand1 \* operand2;

    else if (op == "/")

        return operand1 / operand2;

    else {

        cerr << "Invalid operator: " << op << endl;

        return NAN; // Not a number

    }

}

// Function to evaluate a PostFix expression

double evaluatePostFix(const string& expression) {

    stack<double> operands; // Stack to hold operands

    stringstream ss(expression); // String stream to tokenize the expression

    string token;

    while (ss >> token) { // Tokenize the expression

        if (isNumber(token)) { // If token is a number, push it onto the stack

            operands.push(stod(token));

        } else if (isOperator(token)) { // If token is an operator

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for operator " << token << endl;

                return NAN;

            }

            // Pop the top two operands from the stack

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            // Evaluate the operation and push the result onto the stack

            operands.push(evaluateOperation(operand1, operand2, token));

        } else { // If token is neither a number nor an operator

            cerr << "Invalid token: " << token << endl;

            return NAN;

        }

    }

    // At the end, there should be exactly one operand left in the stack

    if (operands.size() != 1) {

        cerr << "Invalid expression: too many operands" << endl;

        return NAN;

    }

    // Return the final result

    return operands.top();

}

int main() {

    string expression;

    cout << "Enter a PostFix expression: ";

    getline(cin, expression);

    // Evaluate the expression and print the result

    double result = evaluatePostFix(expression);

    if (!isnan(result))

        cout << "Result: " << result << endl;

    return 0;

}

# 2. INKREMENTTI

## Kahden pisteen työn vaatimukset

Toisessa inkrementissä laskimen on toteutettava yhden pisteen vaatimukset ja lisäksi siihen tulee lisää Vaihda-toiminto (komento x (e**x**change)), joka vaihtaa ajon aikana laskimen käyttämässä pinossa kahden ylimmän alkion sisällöt keskenään. Esim:

1 2 x – (laskeekin 2 - 1)

### Toteutus

Inkrementissä 2 ohjelma toteuttaa yksinkertaisen PostFix-laskimen C++-kieltä käyttäen, johon on lisätty "Vaihda" toiminto. Tässä on koodin toiminta vaiheittain:

1. Alustetaan tarvittavat kirjastot: `iostream`, `stack`, `sstream` ja `cmath`. Näitä tarvitaan syötteiden lukemiseen, tietojen tallentamiseen pinoon, merkkijonojen käsittelyyn ja matemaattisten toimintojen suorittamiseen.

2. Määritellään `isOperator`-funktio tarkistamaan, onko annettu merkkijono operaattori (eli +, -, \*, /).

3. Määritellään `isExchange`-funktio tarkistamaan, onko annettu merkkijono "Vaihda" komento.

4. Määritellään `isNumber`-funktio tarkistamaan, onko annettu merkkijono numero.

5. Määritellään `evaluateOperation`-funktio, joka suorittaa annetun operaation kahdelle operandille ja palauttaa tuloksen.

6. Määritellään `evaluatePostFix`-funktio, joka ottaa PostFix-lausekkeen merkkijonona, arvioi sen ja palauttaa lopputuloksen. Tässä funktion osassa lisätään toiminnallisuus "Vaihda" komennolle. Jos lausekkeessa esiintyy "Vaihda" komento, se vaihtaa kahden ylimmän alkion sisällöt pinossa keskenään.

7. Pääohjelmassa pyydetään käyttäjältä syötettä PostFix-lausekkeena.

8. Käyttäjän antama lauseke arvioidaan `evaluatePostFix`-funktiolla.

9. Lopputulos tulostetaan näytölle, jos se on kelvollinen. Jos arviointi epäonnistuu tai lauseke on virheellinen, tulostetaan virheilmoitus.

### Toiminta

Esimerkkejä Vaihda-toiminnon (x) toiminnasta.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Ohjelma toimii myös 1. inkrementin tavoin.

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

Esimerkkejä virheilmoituksista.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

### 2. Inkrementin lähdekoodi

#include <iostream>

#include <stack>

#include <sstream>

#include <cmath>

using namespace std;

// Function to check if a token is an operator

bool isOperator(const string& token) {

    return token == "+" || token == "-" || token == "\*" || token == "/";

}

// Function to check if a token is the "Vaihda" command

bool isExchange(const string& token) {

    return token == "x";

}

bool isNumber(const string& token) {

    stringstream ss(token);

    double num;

    // Try to extract a number from the stringstream

    return ss >> num && ss.eof(); // Return true if extraction successful and no further characters left

}

// Function to evaluate an operation

double evaluateOperation(double operand1, double operand2, const string& op) {

    if (op == "+")

        return operand1 + operand2;

    else if (op == "-")

        return operand1 - operand2;

    else if (op == "\*")

        return operand1 \* operand2;

    else if (op == "/")

        return operand1 / operand2;

    else {

        cerr << "Invalid operator: " << op << endl;

        return NAN; // Not a number

    }

}

// Function to evaluate a PostFix expression

double evaluatePostFix(const string& expression) {

    stack<double> operands; // Stack to hold operands

    stringstream ss(expression); // String stream to tokenize the expression

    string token;

    while (ss >> token) { // Tokenize the expression

        if (isNumber(token)) { // If token is a number, push it onto the stack

            operands.push(stod(token));

        } else if (isOperator(token)) { // If token is an operator

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for operator " << token << endl;

                return NAN;

            }

            // Pop the top two operands from the stack

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            // Evaluate the operation and push the result onto the stack

            operands.push(evaluateOperation(operand1, operand2, token));

        } else if (isExchange(token)) { // If token is the "Vaihda" command

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for 'Vaihda' command" << endl;

                return NAN;

            }

            // Pop the top two operands from the stack and swap them

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            operands.push(operand2);

            operands.push(operand1);

        } else { // If token is neither a number nor an operator nor the "Vaihda" command

            cerr << "Invalid token: " << token << endl;

            return NAN;

        }

    }

    // At the end, there should be exactly one operand left in the stack

    if (operands.size() != 1) {

        cerr << "Invalid expression: too many operands" << endl;

        return NAN;

    }

    // Return the final result

    return operands.top();

}

int main() {

    string expression;

    cout << "Enter a PostFix expression: ";

    getline(cin, expression);

    // Evaluate the expression and print the result

    double result = evaluatePostFix(expression);

    if (!isnan(result))

        cout << "Result: " << result << endl;

    return 0;

}

# 3. INKREMENTTI

## Kolmen pisteen työn vaatimukset

Kolmannessa inkrementissä laskimeen lisätään summatoiminto (komento s (= **s**um)), joka laskee kaikkien pinossa sillä hetkellä olevien lukujen yhteissumman, poistaa tämän jälkeen pinosta summaamiseen käytetyt luvut ja lisää lopuksi yhteissumman pinon huipulle.

Esim.

1 2 3 s

Tämän jälkeen pinossa on vain yksi alkio, jonka arvo on 6.

### Toteutus

1. Kirjastojen sisällyttäminen: Alussa ohjelma sisältää tarvittavat C++ -kirjastot, kuten `iostream`, `stack`, `sstream` ja `cmath`, jotka mahdollistavat syötteen lukemisen, pinojen käytön, merkkijonojen käsittelyn ja matemaattisten toimintojen suorittamisen.

2. Funktio `isOperator`: Tämä funktio tarkistaa, onko annettu merkkijono operaattori (eli +, -, \*, /).

3. Funktio `isExchange`: Tämä funktio tarkistaa, onko annettu merkkijono "Vaihda" komento.

4. Funktio `isSum`: Tämä funktio tarkistaa, onko annettu merkkijono "Summa" komento.

5. Funktio `isNumber`: Tämä funktio tarkistaa, onko annettu merkkijono numero.

6. Funktio `evaluateOperation`: Tämä funktio suorittaa annetun operaation kahdelle operandille ja palauttaa tuloksen.

7. Funktio `evaluatePostFix`: Tämä funktio arvioi annetun PostFix-lausekkeen ja palauttaa lopputuloksen. Se käy läpi lausekkeen merkkijonona, suorittaa operaatiot ja hallitsee vaihto-, summa- ja virheentarkistustoiminnot.

8. `main`-funktio: Tässä pääohjelmassa pyydetään käyttäjältä syöttöä PostFix-lausekkeena. Tämän jälkeen syöte arvioidaan `evaluatePostFix`-funktiolla, ja tulokset tulostetaan näytölle. Jos arviointi epäonnistuu tai lauseke on virheellinen, tulostetaan virheilmoitus.

Näin ohjelma lukee, arvioi ja tulostaa PostFix-lausekkeita käyttäjän antamien komentojen perusteella, mukaan lukien "Vaihda" ja "Summa".

### Toiminta

Esimerkkejä summatoiminnosta.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Koodissa on edelleen samat virheilmoituksen, joten en lähde tässä osiossa niitä erittelemään. Ohjelma myös toimii myös aikaisemmilla menetelmillä, koska 3. inkrementin toiminta on vain lisätty edelliseen ohjelmakoodiin.

### 3. Inkrementin lähdekoodi

#include <iostream>

#include <stack>

#include <sstream>

#include <cmath>

using namespace std;

// Function to check if a token is an operator

bool isOperator(const string& token) {

    return token == "+" || token == "-" || token == "\*" || token == "/";

}

// Function to check if a token is the "Vaihda" command

bool isExchange(const string& token) {

    return token == "x";

}

// Function to check if a token is the "Summa" command

bool isSum(const string& token) {

    return token == "s";

}

bool isNumber(const string& token) {

    stringstream ss(token);

    double num;

    // Try to extract a number from the stringstream

    return ss >> num && ss.eof(); // Return true if extraction successful and no further characters left

}

// Function to evaluate an operation

double evaluateOperation(double operand1, double operand2, const string& op) {

    if (op == "+")

        return operand1 + operand2;

    else if (op == "-")

        return operand1 - operand2;

    else if (op == "\*")

        return operand1 \* operand2;

    else if (op == "/")

        return operand1 / operand2;

    else {

        cerr << "Invalid operator: " << op << endl;

        return NAN; // Not a number

    }

}

// Function to evaluate a PostFix expression

double evaluatePostFix(const string& expression) {

    stack<double> operands; // Stack to hold operands

    stringstream ss(expression); // String stream to tokenize the expression

    string token;

    while (ss >> token) { // Tokenize the expression

        if (isNumber(token)) { // If token is a number, push it onto the stack

            operands.push(stod(token));

        } else if (isOperator(token)) { // If token is an operator

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for operator " << token << endl;

                return NAN;

            }

            // Pop the top two operands from the stack

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            // Evaluate the operation and push the result onto the stack

            operands.push(evaluateOperation(operand1, operand2, token));

        } else if (isExchange(token)) { // If token is the "Vaihda" command

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for 'Vaihda' command" << endl;

                return NAN;

            }

            // Pop the top two operands from the stack and swap them

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            operands.push(operand2);

            operands.push(operand1);

        } else if (isSum(token)) { // If token is the "Summa" command

            double sum = 0;

            while (!operands.empty()) {

                sum += operands.top();

                operands.pop();

            }

            operands.push(sum);

        } else { // If token is neither a number nor an operator nor the "Vaihda" command nor the "Summa" command

            cerr << "Invalid token: " << token << endl;

            return NAN;

        }

    }

    // At the end, there should be exactly one operand left in the stack

    if (operands.size() != 1) {

        cerr << "Invalid expression: too many operands" << endl;

        return NAN;

    }

    // Return the final result

    return operands.top();

}

int main() {

    string expression;

    cout << "Enter a PostFix expression: ";

    getline(cin, expression);

    // Evaluate the expression and print the result

    double result = evaluatePostFix(expression);

    if (!isnan(result))

        cout << "Result: " << result << endl;

    return 0;

}

# 4. INKREMENTTI

## Neljän pisteen työn vaatimukset

Neljännessä inkrementissä toteutetaan laskimeen keskiarvotoiminto, joka laskee kaikkien pinossa olevien lukujen keskiarvon. Keskiarvolaskentaan osallistuvat luvut poistuvat pinosta ja tulos (keskiarvo) painuu pinon päälle. Tämä laskutoimitus nimetään kirjaimella a (= **a**verage).

### Toteutus

1. Tarvittavien kirjastojen sisällyttäminen: Aluksi sisällytämme tarvittavat C++-kirjastot, kuten `iostream`, `stack`, `sstream` ja `cmath`.

2. Funktio `isOperator`: Tämä funktio tarkistaa, onko annettu merkkijono operaattori (eli `+`, `-`, `\*` tai `/`). Se palauttaa totuusarvon sen perusteella, onko annettu merkkijono operaattori vai ei.

3. Funktio `isExchange`: Tämä funktio tarkistaa, onko annettu merkkijono "Vaihda" -komento, joka vaihtaa pinon kahden ylimmän alkion sisällöt keskenään.

4. Funktio `isSum`: Tämä funktio tarkistaa, onko annettu merkkijono "Summa" -komento, joka laskee kaikkien pinossa olevien lukujen yhteissumman, poistaa tämän jälkeen pinosta summaamiseen käytetyt luvut ja lisää lopuksi yhteissumman pinon huipulle.

5. Funktio `isAverage`: Tämä funktio tarkistaa, onko annettu merkkijono "Keskiarvo" -komento, joka laskee kaikkien pinossa olevien lukujen keskiarvon.

6. Funktio `isNumber`: Tämä funktio tarkistaa, onko annettu merkkijono numero. Se käyttää `stringstream`-luokkaa muuntaakseen merkkijonon numeroksi ja tarkistaa sitten, onko muunnos onnistunut ja onko lisäksi kaikki merkit käytetty.

7. Funktio `evaluateOperation`: Tämä funktio arvioi kahden operandin ja operaattorin tuloksen. Se ottaa kaksi lukua ja operaattorin merkkijonona, suorittaa operaation ja palauttaa tuloksen.

8. Funktio `evaluatePostFix`: Tämä funktio arvioi annetun PostFix-lausekkeen. Se käy läpi lausekkeen, merkki kerrallaan, ja suorittaa tarvittavat toiminnot kullekin merkille. Se käyttää `stack`-tietorakennetta pitääkseen kirjaa luvuista ja operaattoreista.

9. Pääohjelma `main`: Pääohjelma lukee käyttäjältä PostFix-lausekkeen, kutsuu `evaluatePostFix`-funktiota arvioimaan sen ja tulostaa lopputuloksen.

Ohjelma on siis rakennettu tässä vaiheessa käsittämään neljää erilaista toimintoa, jotka voivat esiintyä PostFix-lausekkeissa, ja suorittaa ne oikein.

### Toiminta

Esimerkkejä keskiarvokomennon (a) toiminnasta.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Koodissa on edelleen samat virheilmoitukset, joita en lähde tässäkään osiossa erittelemään. Ohjelma myös toimii myös aikaisemmilla menetelmillä, koska 4. inkrementin toiminta on vain lisätty edelliseen ohjelmakoodiin.

### 4. Inkrementin lähdekoodi

#include <iostream>

#include <stack>

#include <sstream>

#include <cmath>

using namespace std;

// Function to check if a token is an operator

bool isOperator(const string& token) {

    return token == "+" || token == "-" || token == "\*" || token == "/";

}

// Function to check if a token is the "Vaihda" command

bool isExchange(const string& token) {

    return token == "x";

}

// Function to check if a token is the "Summa" command

bool isSum(const string& token) {

    return token == "s";

}

// Function to check if a token is the "Keskiarvo" command

bool isAverage(const string& token) {

    return token == "a";

}

bool isNumber(const string& token) {

    stringstream ss(token);

    double num;

    // Try to extract a number from the stringstream

    return ss >> num && ss.eof(); // Return true if extraction successful and no further characters left

}

// Function to evaluate an operation

double evaluateOperation(double operand1, double operand2, const string& op) {

    if (op == "+")

        return operand1 + operand2;

    else if (op == "-")

        return operand1 - operand2;

    else if (op == "\*")

        return operand1 \* operand2;

    else if (op == "/")

        return operand1 / operand2;

    else {

        cerr << "Invalid operator: " << op << endl;

        return NAN; // Not a number

    }

}

// Function to evaluate a PostFix expression

double evaluatePostFix(const string& expression) {

    stack<double> operands; // Stack to hold operands

    stringstream ss(expression); // String stream to tokenize the expression

    string token;

    while (ss >> token) { // Tokenize the expression

        if (isNumber(token)) { // If token is a number, push it onto the stack

            operands.push(stod(token));

        } else if (isOperator(token)) { // If token is an operator

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for operator " << token << endl;

                return NAN;

            }

            // Pop the top two operands from the stack

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            // Evaluate the operation and push the result onto the stack

            operands.push(evaluateOperation(operand1, operand2, token));

        } else if (isExchange(token)) { // If token is the "Vaihda" command

            if (operands.size() < 2) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for 'Vaihda' command" << endl;

                return NAN;

            }

            // Pop the top two operands from the stack and swap them

            double operand2 = operands.top();

            operands.pop();

            double operand1 = operands.top();

            operands.pop();

            operands.push(operand2);

            operands.push(operand1);

        } else if (isSum(token)) { // If token is the "Summa" command

            double sum = 0;

            while (!operands.empty()) {

                sum += operands.top();

                operands.pop();

            }

            operands.push(sum);

        } else if (isAverage(token)) { // If token is the "Keskiarvo" command

            if (operands.size() < 1) { // Ensure there are enough operands in the stack

                cerr << "Invalid expression: insufficient operands for 'Keskiarvo' command" << endl;

                return NAN;

            }

            // Calculate the average of all numbers in the stack

            double sum = 0;

            int count = 0;

            while (!operands.empty()) {

                sum += operands.top();

                operands.pop();

                count++;

            }

            // Push the average onto the stack

            operands.push(sum / count);

        } else { // If token is neither a number nor an operator nor the "Vaihda" command nor the "Summa" command nor the "Keskiarvo" command

            cerr << "Invalid token: " << token << endl;

            return NAN;

        }

    }

    // At the end, there should be exactly one operand left in the stack

    if (operands.size() != 1) {

        cerr << "Invalid expression: too many operands" << endl;

        return NAN;

    }

    // Return the final result

    return operands.top();

}

int main() {

    string expression;

    cout << "Enter a PostFix expression: ";

    getline(cin, expression);

    // Evaluate the expression and print the result

    double result = evaluatePostFix(expression);

    if (!isnan(result))

        cout << "Result: " << result << endl;

    return 0;

}

# 5. INKREMENTTI

## Viiden pisteen työn vaatimukset

Tee laskimeesi Unix-/Linux-laitteilta löytyvän dc-ohjelman kaltainen käyttöliittymä, joka ei tulosta mitään kehotteita vaan ohjelmalle annetaan suoraan haluttu laskulauseke postfix-muodossa ja ohjelma tulostaa sen arvon (ks. http://en.wikipedia.org/wiki/Dc\_(computer\_program)). Esimerkki:

1 3 + 7.5 –

-3.5

Viiden pisteen laskimessa tulee olla seuraavat laskutoimitukset: yhteenlasku (+), vähennyslasku (-), kertolasku (\*), jakolasku (/), jakojäännösoperaatio (%), potenssiin korotus (^), neliöjuuri (v) sekä edellä vaaditut laskutoimitukset 1 – 4 pisteen arvoisissa töissä.

Koska mitään aiemmin tehtyä ei oteta pois laskinohjelmasta sisällytä kehotteiden tulostaminen ohjelmaan, jos se käynnistetään tietyllä komentorivioptiolla; esim. laskin.exe –p

(= prompt).

### Toteutus

Alustus:

Tarvittavien kirjastojen sisällyttäminen ja nimiavaruuden std määrittäminen.

Operaattorien ja numeroiden tunnistaminen:

Merkkijonon tarkistus operaattoriksi (isOperator) ja numeroksi (isNumber).

Operaatioiden arviointi:

Operaatioiden suorittaminen kahdella operandilla (evaluateOperation).

Postfix-lausekkeen arviointi:

PostFix-lausekkeen arviointi käyttämällä pinoa (stack) operandien säilyttämiseen ja operaatioiden suorittamiseen (evaluatePostFix).

Pääohjelma (main):

Ohjelman käynnistyksen tarkistus -p-lipulla.

PostFix-lausekkeen pyytäminen käyttäjältä (getline) tai sen lukeminen suoraan syötteestä.

Arvioinnin suorittaminen annetulle lausekkeelle ja tuloksen tulostaminen.

Lisäominaisuudet:

Vaihda-toiminnon (x), summa-toiminnon (s), keskiarvotoiminnon (a) ja kaikkien aiemmin vaadittujen toimintojen toteutus.

### Toiminta

Esimerkkejä toiminnasta:

1. Peruslaskutoimitukset

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

1. Kompleksiset laskutoimitukset

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus, Grafiikka

Kuvaus luotu automaattisesti

1. Jakojäännösoperaatio:

Kuva, joka sisältää kohteen Fontti, kuvakaappaus, teksti

Kuvaus luotu automaattisesti

1. Potenssiin korotus

Kuva, joka sisältää kohteen Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

1. Neliöjuuri

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus, Grafiikka

Kuvaus luotu automaattisesti

1. Virhesyötteet

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Aikaisempien inkrementtien toiminta testattuna prompt (-p) toiminnolla.

Exchange (x), sum (s), average (a):

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Ohjelmalla pystyy lukemaan myös tiedostosta laskutoimituksen ja ohjelma tuo sen komentoriville.

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

### 5. inkrementin lähdekoodi

#include <iostream>

#include <stack>

#include <sstream>

#include <cmath>

using namespace std;

// Function to check if a token is an operator

bool isOperator(const string& token) {

    return token == "+" || token == "-" || token == "\*" || token == "/" || token == "%" || token == "^" || token == "v" || token == "s" || token == "a" || token == "x";

}

bool isNumber(const string& token) {

    stringstream ss(token);

    double num;

    // Try to extract a number from the stringstream

    return ss >> num && ss.eof(); // Return true if extraction successful and no further characters left

}

// Function to evaluate an operation

double evaluateOperation(double operand1, double operand2, const string& op) {

    if (op == "+")

        return operand1 + operand2;

    else if (op == "-")

        return operand1 - operand2;

    else if (op == "\*")

        return operand1 \* operand2;

    else if (op == "/")

        return operand1 / operand2;

    else if (op == "%")

        return fmod(operand1, operand2);

    else if (op == "^")

        return pow(operand1, operand2);

    else if (op == "v") // Square root operation

        return sqrt(operand1);

    else if (op == "s") { // Sum operation

        return operand1 + operand2;

    } else if (op == "a") { // Average operation

        return (operand1 + operand2) / 2.0;

    } else if (op == "x") { // Exchange operation

        return operand2; // Return the second operand (the top of the stack)

    } else {

        cerr << "Invalid operator: " << op << endl;

        return NAN; // Not a number

    }

}

// Function to evaluate a PostFix expression

double evaluatePostFix(const string& expression) {

    stack<double> operands; // Stack to hold operands

    stringstream ss(expression); // String stream to tokenize the expression

    string token;

    while (ss >> token) { // Tokenize the expression

        if (isNumber(token)) { // If token is a number, push it onto the stack

            operands.push(stod(token));

        } else if (isOperator(token)) { // If token is an operator

            if (token == "s" || token == "a") { // Handle sum and average operations differently

                double sum = 0;

                while (!operands.empty()) { // Iterate through all elements in the stack

                    sum += operands.top(); // Add the top element to the sum

                    operands.pop(); // Remove the top element

                }

                operands.push(sum); // Push the sum onto the stack

            } else if (token == "x") { // If token is the "Vaihda" command

                if (operands.size() < 2) { // Ensure there are enough operands in the stack

                    cerr << "Invalid expression: insufficient operands for 'Vaihda' command" << endl;

                    return NAN;

                }

                // Pop the top two operands from the stack and swap them

                double operand2 = operands.top();

                operands.pop();

                double operand1 = operands.top();

                operands.pop();

                operands.push(operand2);

                operands.push(operand1);

            } else { // If token is a regular operator

                if (operands.size() < 2) { // Ensure there are enough operands in the stack

                    cerr << "Invalid expression: insufficient operands for operator " << token << endl;

                    return NAN;

                }

                // Pop the top two operands from the stack

                double operand2 = operands.top();

                operands.pop();

                double operand1 = operands.top();

                operands.pop();

                // Evaluate the operation and push the result onto the stack

                operands.push(evaluateOperation(operand1, operand2, token));

            }

        } else { // If token is neither a number nor an operator

            cerr << "Invalid token: " << token << endl;

            return NAN;

        }

    }

    // At the end, there should be exactly one operand left in the stack

    if (operands.size() != 1) {

        cerr << "Invalid expression: too many operands" << endl;

        return NAN;

    }

    // Return the final result

    return operands.top();

}

int main(int argc, char \*argv[]) {

    // Check if the program was invoked with the -p flag for prompt

    bool prompt = false;

    if (argc > 1 && string(argv[1]) == "-p") {

        prompt = true;

    }

    string expression;

    if (prompt) {

        // Print prompt

        cout << "Enter a PostFix expression: ";

        getline(cin, expression);

    } else {

        // Read the expression directly

        getline(cin, expression);

    }

    // Evaluate the expression and print the result

    double result = evaluatePostFix(expression);

    if (!isnan(result))

        cout << result << endl;

    return 0;

}

# POHDINTA

Projekti oli mielenkiintoinen ja onnistui yllättävän sujuvasti. Mielestäni onnistuin projektissa hyvin ja kaikki inkrementit toimivat tehtävänannon mukaisesti.

Tein projektin viidellä inkrementillä ja täten uskon sen olevan myös viiden arvosanapisteen arvoinen.

Tuntikirjanpito

|  |  |  |
| --- | --- | --- |
| Päivämäärä | Kulunut aika | Mitä tehtiin? |
| 29.01.2024 | 1,5 h | 1. inkrementti testauksineen |
| 31.01.2024 | 2,5 h | Dokumentointi 1. inkrementistä ja 2. inkrementti |
| 01.02.2024 | 2 h | Dokumentointi 2. inkrementistä ja 3. inkrementti |
| 06.02.2024 | 3 h | 3. inkrementti ja 4 inkrementti loppuun sekä niiden dokumentointi |
| 07.02.2024 | 3 h | 5. inkrementti melkein kokonaisuudessaan ja dokumentoinnin aloitus |
| 08.02.2024 | 4 h | Testaukset, 5 inkrementin korjaus siihen kuntoon, että kaikki aikaisemmatkin inkrementit toimivat. Lopullinen dokumentoinnin tarkistus ja päivitys. |