**PENJELASAN UTS NO. 3**

Salsabila Nur Aqmarina Aulia

2209076041

UTS Struktur Data

Pada soal ini meminta kita untuk membuat sebuah program kalkulator notasi postfix dengan menggunakan struktur data stack. Berikut adalah penjelasan tentang setiap bagian program:

**1. Membuat Class `Stack` dengan Operasi `push`, `pop`, dan `isEmpty`**

Class `Stack` ini akan digunakan untuk menyimpan dan mengambil *operand* saat melakukan evaluasi ekspresi postfix. Tiga operasi yang dibutuhkan adalah:

- \*\*`push`\*\*: Menambah elemen ke atas stack.

- \*\*`pop`\*\*: Mengambil dan menghapus elemen teratas dari stack.

- \*\*`isEmpty`\*\*: Memeriksa apakah stack kosong.

Contoh implementasi class `Stack` dalam C++:

#include <iostream>

#include <vector>

class Stack {

private:

std::vector<double> data;

public:

void push(double value) {

data.push\_back(value);

}

double pop() {

if (!isEmpty()) {

double value = data.back();

data.pop\_back();

return value;

} else {

throw std::runtime\_error("Stack kosong!");

}

}

bool isEmpty() const {

return data.empty();

}

};

```

**2. Mengimplementasikan Fungsi untuk Mengevaluasi Ekspresi Postfix**

Untuk mengevaluasi ekspresi postfix, program akan menggunakan `Stack`. Setiap kali menemukan operand, operand tersebut dimasukkan (di-push) ke stack. Ketika menemukan operator, dua operand teratas pada stack diambil (pop), kemudian operasi dilakukan sesuai operator, dan hasilnya dikembalikan ke stack. Pada akhir ekspresi, hasil akhir dari perhitungan ada pada elemen teratas stack.

Pada C++ :

double evaluasiPostfix(const std::string &expresi) {

Stack stack;

for (size\_t i = 0; i < expresi.length(); ++i) {

char token = expresi[i];

if (isdigit(token)) {

stack.push(token - '0'); // Konversi karakter digit ke angka

} else if (token == '+' || token == '-' || token == '\*' || token == '/' || token == '^') {

double operand2 = stack.pop();

double operand1 = stack.pop();

double result = 0;

switch (token) {

case '+': result = operand1 + operand2; break;

case '-': result = operand1 - operand2; break;

case '\*': result = operand1 \* operand2; break;

case '/': result = operand1 / operand2; break;

case '^': result = pow(operand1, operand2); break;

}

stack.push(result); // Push hasil operasi ke stack

}

}

return stack.pop(); // Hasil akhir

}

```

**3. Menangani Operasi Aritmatika Dasar (+, -, \*, /) dan Operator Pangkat (^)**

Pada fungsi `evaluasiPostfix`, kita menggunakan pernyataan `switch` untuk melakukan operasi aritmatika dasar. Operator `^` untuk pangkat membutuhkan fungsi `pow` dari pustaka `cmath` di C++. Setiap kali operator ditemukan, dua operand diambil dari stack, operasi dilakukan, dan hasilnya dikembalikan ke stack.

Pada C++ :

case '+': result = operand1 + operand2; break;

case '-': result = operand1 - operand2; break;

case '\*': result = operand1 \* operand2; break;

case '/': result = operand1 / operand2; break;

case '^': result = pow(operand1, operand2); break;

```

**4. Menampilkan Langkah-Langkah Evaluasi Ekspresi Postfix**

Untuk menampilkan langkah-langkah proses evaluasi, kita dapat menambahkan kode untuk menampilkan isi stack setelah setiap operasi. Setiap kali `push` atau `pop` dilakukan, program akan mencetak isi stack. Ini berguna untuk melihat proses evaluasi langkah demi langkah.

Pada C++ :

double evaluasiPostfix(const std::string &expresi) {

Stack stack;

for (size\_t i = 0; i < expresi.length(); ++i) {

char token = expresi[i];

if (isdigit(token)) {

double value = token - '0';

stack.push(value);

std::cout << "Push " << value << " ke stack\n";

} else if (token == '+' || token == '-' || token == '\*' || token == '/' || token == '^') {

double operand2 = stack.pop();

double operand1 = stack.pop();

double result = 0;

switch (token) {

case '+': result = operand1 + operand2; break;

case '-': result = operand1 - operand2; break;

case '\*': result = operand1 \* operand2; break;

case '/': result = operand1 / operand2; break;

case '^': result = pow(operand1, operand2); break;

}

std::cout << "Operasi: " << operand1 << " " << token << " " << operand2 << " = " << result << "\n";

stack.push(result);

}

}

double finalResult = stack.pop();

std::cout << "Hasil akhir: " << finalResult << "\n";

return finalResult;

}

``’`

**5. Program Utama**

Program utama meminta input ekspresi postfix dari pengguna, kemudian menjalankan fungsi `evaluasiPostfix` untuk menghitung hasilnya. Hasil dan langkah-langkah evaluasi akan ditampilkan.

Pada C++:

int main() {

std::string expresi;

std::cout << "Masukkan ekspresi postfix (contoh: 53+82-\*): ";

std::cin >> expresi;

try {

double hasil = evaluasiPostfix(expresi);

std::cout << "Hasil dari ekspresi postfix: " << hasil << "\n";

} catch (const std::exception &e) {

std::cout << "Error: " << e.what() << "\n";

}

return 0;

}

```

Jadi pada program ini kita dapat melakukan kalkulasi notasi postfix dengan *stack*, menunjukkan setiap langkah evaluasi, dan menangani operasi aritmatika serta pangkat. Dengan implementasi ini, kita dapat melihat setiap operasi yang dilakukan di *stack*, memberikan transparansi pada proses perhitungan *postfix*.