**Laporan Praktikum**

**Algoritma Struktur Data**

Stack

**Oleh:**

Oscar Javier Abdullah/ 5223600001

**Program Studi STr Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika**

**Negeri Surabaya**

**2024**

**Megimplementasikan Stack Dengan Menggunakan Linked List**

* Push: Menambahkan elemen baru ke dalam stack, yaitu dengan menambahkan elemen baru di awal linked list.
* Pop: Menghapus elemen teratas dari stack, yaitu dengan menghapus elemen pertama dari linked list.
* Peek: Melihat elemen teratas dari stack, yaitu dengan melihat elemen pertama dari linked list.
* Display: Menampilkan seluruh elemen dalam stack, yaitu dengan menampilkan seluruh elemen dalam linked list.

Code Sebagai Berikut :

#include <iostream>

class ListStack {

private:

struct Node {

int value;

Node\* next;

Node(int v, Node\* n) : value(v), next(n) {}

};

Node\* head = nullptr;

int count = 0;

public:

int size() {

return count;

}

bool Empty() {

return count == 0;

}

int peek() {

if (Empty()) {

throw std::runtime\_error("ListStackEmptyException");

}

return head->value;

}

void Push(int value) {

head = new Node(value, head);

count++;

}

int Pop() {

if (Empty()) {

throw std::runtime\_error("ListStackEmptyException");

}

int value = head->value;

Node\* temp = head;

head = head->next;

delete temp;

count--;

return value;

}

void insertAtBottom(int value) {

if (Empty()) {

Push(value);

} else {

int temp = Pop();

insertAtBottom(value);

Push(temp);

}

}

void print() {

Node\* temp = head;

while (temp != nullptr) {

std::cout << temp->value << " ";

temp = temp->next;

}

}

};

int main() {

ListStack s;

for (int i = 1; i <= 100; i++)

s.Push(i);

for (int i = 1; i <= 70; i++)

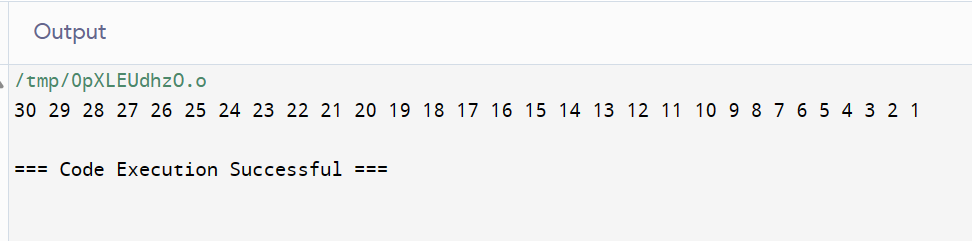
s.Pop();

s.print();

return 0;

}

Hasil / Output akan seperti :



**Keseimbangan Tanda Kurung**

Pada fungsi isBalancedParenthesis(), kita menggunakan **stack** untuk melacak tanda kurung yang belum ditutup. Jika tanda kurung seimbang, maka fungsi akan mengembalikan true, dan jika tidak, akan mengembalikan false.

#include <iostream>

#include <stack>

#include <string>

bool isBalancedParenthesis(const std::string& expression) {

std::stack<char> parenStack;

for (char ch : expression) {

if (ch == '(' || ch == '[' || ch == '{') {

// Push opening parentheses onto the stack

parenStack.push(ch);

} else if (ch == ')' || ch == ']' || ch == '}') {

// Check if the stack is empty or if the top element doesn't match the closing parenthesis

if (parenStack.empty() ||

(ch == ')' && parenStack.top() != '(') ||

(ch == ']' && parenStack.top() != '[') ||

(ch == '}' && parenStack.top() != '{')) {

return false; // Unbalanced

}

parenStack.pop(); // Pop the matching opening parenthesis

}

}

// If the stack is empty, all parentheses are balanced

return parenStack.empty();

}

int main() {

std::string input;

std::cout << "Masukkan ekspresi dengan tanda kurung: ";

std::cin >> input;

if (isBalancedParenthesis(input)) {

std::cout << "Tanda kurung seimbang." << std::endl;

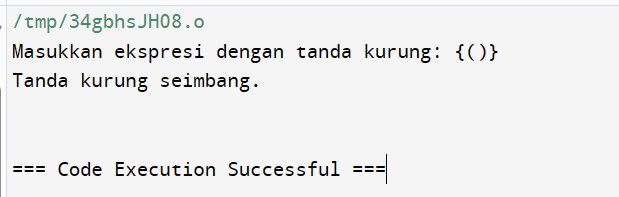
} else {

std::cout << "Tanda kurung tidak seimbang." << std::endl;

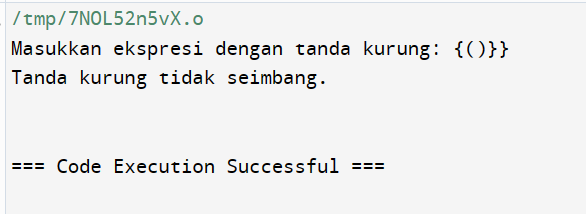
}

return 0;

}

Hasil Output Seimbang :

Hasil Output Tidak Seimbang :



**Konversi Infix menjadi Postfix**

Fungsi ini mengonversi ekspresi matematika dalam notasi infix menjadi notasi postfix.

**Cara kerjanya:**

* Fungsi menggunakan stack untuk mengatur operator dan operand.
* Ketika karakter dalam ekspresi adalah operand (misalnya huruf atau angka), fungsi langsung menambahkannya ke dalam string hasil (notasi postfix).
* Ketika karakter adalah operator (misalnya ‘+’, ‘-’, ‘\*’, atau ‘/’), fungsi membandingkan prioritas operator dan mengatur ulang operator pada stack hingga urutan postfix terbentuk.
* Setelah semua karakter diperiksa, sisa operator pada stack dipindahkan ke hasil.

Code

#include <iostream>

#include <stack>

#include <string>

using namespace std;

// Fungsi untuk memeriksa tingkat prioritas operator

int precedence(char op) {

if (op == '+' || op == '-')

return 1;

if (op == '\*' || op == '/')

return 2;

return 0;

}

// Fungsi untuk mengonversi infix menjadi postfix

string infixToPostfix(const string& infix) {

stack<char> operatorStack;

string postfix = "";

for (char ch : infix) {

if (isalnum(ch)) {

// Jika karakter adalah operand, tambahkan ke postfix

postfix += ch;

} else if (ch == '(') {

// Jika karakter adalah '(', push ke stack

operatorStack.push(ch);

} else if (ch == ')') {

// Jika karakter adalah ')', pop dari stack hingga '(' ditemukan

while (!operatorStack.empty() && operatorStack.top() != '(') {

postfix += operatorStack.top();

operatorStack.pop();

}

operatorStack.pop(); // Hapus '(' dari stack

} else {

// Jika karakter adalah operator, pop dari stack hingga prioritas lebih rendah

while (!operatorStack.empty() && precedence(ch) <= precedence(operatorStack.top())) {

postfix += operatorStack.top();

operatorStack.pop();

}

operatorStack.push(ch); // Push operator ke stack

}

}

// Pop sisa operator dari stack

while (!operatorStack.empty()) {

postfix += operatorStack.top();

operatorStack.pop();

}

return postfix;

}

int main() {

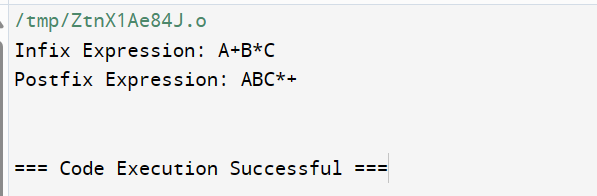
string infixExpression = "A+B\*C";

cout << "Infix Expression: " << infixExpression << endl;

cout << "Postfix Expression: " << infixToPostfix(infixExpression) << endl;

return 0;

}

Hasil Output :

**Konversi Infix Menjadi Prefix**

#include <iostream>

#include <stack>

#include <string>

#include <algorithm>

using namespace std;

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

int getPriority(char op) {

if (op == '-' || op == '+')

return 1;

else if (op == '\*' || op == '/')

return 2;

return 0;

}

string infixToPrefix(const string& infix) {

stack<char> operators;

string prefix = "";

for (int i = infix.length() - 1; i >= 0; i--) {

char ch = infix[i];

if (isalnum(ch)) {

// Operand: directly add to prefix

prefix += ch;

} else if (ch == ')') {

// Closing parenthesis: push to stack

operators.push(ch);

} else if (ch == '(') {

// Opening parenthesis: pop from stack until closing parenthesis

while (!operators.empty() && operators.top() != ')') {

prefix += operators.top();

operators.pop();

}

operators.pop(); // Remove the closing parenthesis

} else if (isOperator(ch)) {

// Operator: pop from stack until lower priority or empty

while (!operators.empty() && getPriority(ch) < getPriority(operators.top())) {

prefix += operators.top();

operators.pop();

}

operators.push(ch);

}

}

// Pop remaining operators from stack

while (!operators.empty()) {

prefix += operators.top();

operators.pop();

}

// Reverse the prefix expression

reverse(prefix.begin(), prefix.end());

return prefix;

}

int main() {

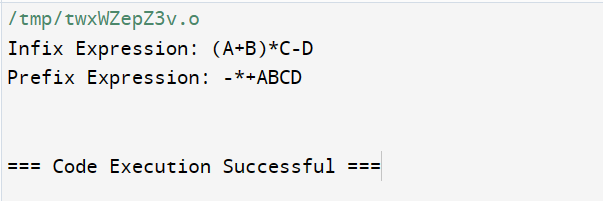
string infixExpression = "(A+B)\*C-D";

cout << "Infix Expression: " << infixExpression << endl;

cout << "Prefix Expression: " << infixToPrefix(infixExpression) << endl;

return 0;

}

Hasil Output :

**Evaluasi Postfix**

* Fungsi postfixEvaluate menerima ekspresi postfix sebagai string dan mengembalikan hasil evaluasi.
* Ekspresi postfix harus memiliki format yang benar, dengan angka dan operator dipisahkan oleh spasi.
* Operator yang didukung: +, -, \*, /, dan ^ (pangkat).
* Pastikan memasukkan ekspresi postfix yang valid saat menjalankan program ini.

Code

#include <iostream>

#include <stack>

#include <string>

#include <cctype>

using namespace std;

int evaluatePostfix(const string& postfix) {

stack<int> stack;

for (char ch : postfix) {

if (isdigit(ch)) {

stack.push(ch - '0');

}

else {

int operand2 = stack.top();

stack.pop();

int operand1 = stack.top();

stack.pop();

switch (ch) {

case '+':

stack.push(operand1 + operand2);

break;

case '-':

stack.push(operand1 - operand2);

break;

case '\*':

stack.push(operand1 \* operand2);

break;

case '/':

stack.push(operand1 / operand2);

break;

}

}

}

return stack.top();

}

int main() {

string postfix = "40+66\*-";

int result = evaluatePostfix(postfix);

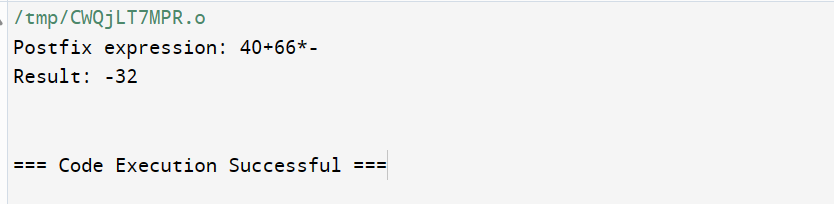
cout << "Postfix expression: " << postfix << endl;

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

return 0;

}

Hasil Output :



**Palindrome string dengan Stack**

* Fungsi isPalindrome memeriksa apakah string yang diberikan merupakan palindrome.
* String dianggap palindrome jika huruf-hurufnya dapat dibaca sama baik dari depan maupun belakang.
* Implementasi ini menggunakan stack untuk membandingkan karakter-karakter pada posisi yang sesuai.

Code

#include <iostream>

#include <stack>

#include <string>

using namespace std;

// Fungsi untuk memeriksa apakah string merupakan palindrome

bool isPalindrome(const string& s) {

int length = s.size();

stack<char> charStack;

int mid = length / 2;

// Push karakter-karakter hingga pertengahan string ke dalam stack

for (int i = 0; i < mid; ++i) {

charStack.push(s[i]);

}

// Jika panjang string ganjil, abaikan karakter tengah

if (length % 2 != 0) {

++mid;

}

// Bandingkan karakter-karakter dari tengah hingga akhir string

for (int i = mid; i < length; ++i) {

char ele = charStack.top();

charStack.pop();

if (ele != s[i]) {

return false;

}

}

return true;

}

int main() {

string input;

cout << "Masukkan kata: ";

cin >> input;

if (isPalindrome(input)) {

cout << "Ya, itu adalah palindrome." << endl;

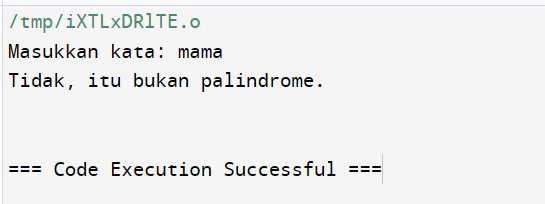
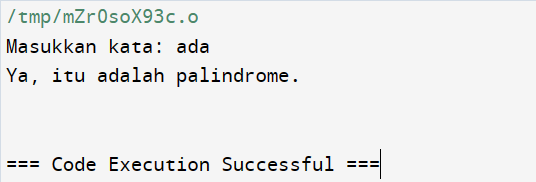
} else {

cout << "Tidak, itu bukan palindrome." << endl;

}

return 0;

}

Hasil output baik palindrome, ataupun bukan :