

# **Praktikum Algoritma dan Struktur Data**

## **Overview Stack**



**Oleh :**

Rayhan Elmo Athalah Saputra (5223600027)

**Program Studi Sarjana Terapan Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika Negeri Surabaya**

**2024**

## 1. Implementasikan Stack Menggunakan Linked List

Kelas `ListStack` merupakan implementasi Stack, dengan menggunakan operasi dasar seperti `push` (menambahkan elemen ke Stack), `pop` (menghapus elemen dari Stack), `peek` (melihat elemen teratas tanpa menghapusnya), `size` (mendapatkan ukuran Stack), dan `empty` (memeriksa apakah Stack kosong).

```
#include <iostream>
using namespace std;

// Deklarasi kelas Node untuk merepresentasikan simpul dalam tumpukan
class Node {
public:
    int value; // Nilai dari simpul
    Node* next; // Pointer ke simpul berikutnya

    // Konstruktor untuk inisialisasi nilai dan pointer
    Node(int val, Node* n = nullptr) {
        value = val;
        next = n;
    }
};

// Deklarasi kelas ListStack untuk merepresentasikan tumpukan
menggunakan linked list
class ListStack {
private:
    Node* head; // Pointer ke elemen paling atas (head) dari tumpukan
    int count; // Jumlah elemen dalam tumpukan

public:
    // Konstruktor untuk inisialisasi tumpukan kosong
    ListStack() {
        head = nullptr;
        count = 0;
    }

    // Fungsi untuk mengembalikan jumlah elemen dalam tumpukan
    int size() {
        return count;
    }

    // Fungsi untuk memeriksa apakah tumpukan kosong atau tidak
    bool empty() {
        return count == 0;
    }
}
```

// Fungsi untuk melihat nilai dari elemen paling atas (head) tanpa menghapusnya

```
int peek() {  
    if (empty()) {  
        throw out_of_range("Stack is empty");  
    }  
    return head->value;  
}
```

// Fungsi untuk menambahkan elemen baru ke paling atas (head) dari tumpukan

```
void push(int value) {  
    head = new Node(value, head);  
    count++;  
}
```

// Fungsi untuk menghapus dan mengembalikan nilai dari elemen paling atas (head) dari tumpukan

```
int pop() {  
    if (empty()) {  
        throw out_of_range("Stack is empty");  
    }  
    int value = head->value;  
    Node* temp = head;  
    head = head->next;  
    delete temp;  
    count--;  
    return value;  
}
```

};

// Fungsi utama program

```
int main() {  
    ListStack stack; // Membuat objek tumpukan dari kelas ListStack  
    stack.push(1); // Menambahkan elemen 1 ke tumpukan  
    stack.push(2); // Menambahkan elemen 2 ke tumpukan  
    stack.push(3); // Menambahkan elemen 3 ke tumpukan  
  
    // Menampilkan nilai dari elemen paling atas (top) dari tumpukan  
    cout << "Top element: " << stack.peek() << endl;  
    // Menampilkan jumlah elemen dalam tumpukan  
    cout << "Size of stack: " << stack.size() << endl;  
  
    stack.pop(); // Menghapus elemen paling atas (top) dari tumpukan  
    // Menampilkan nilai dari elemen paling atas (top) setelah penghapusan  
    cout << "Top element after pop: " << stack.peek() << endl;
```

```

    return 0;
}

```

The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` defines a `Node` class with `value` and `next` attributes, and a `ListStack` class. The output window shows the results of a program execution: `Top element: 3`, `Size of stack: 3`, and `Top element after pop: 2`, followed by `=== Code Execution Successful ===`.

## 2. Memeriksa keseimbangan Tanda Kurung

Memeriksa keseimbangan tanda kurung dalam sebuah ekspresi menggunakan Stack dengan mendorong tanda kurung pembuka ke dalam Stack dan mencocokkannya dengan tanda kurung penutup.

```

#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Fungsi untuk memeriksa apakah urutan tanda kurung dalam ekspresi
seimbang atau tidak
bool isBalancedParenthesis(const string& expr) {
    stack<char> stack; // Membuat tumpukan untuk menyimpan tanda
    kurung

    // Melakukan iterasi pada setiap karakter dalam ekspresi
    for (char ch : expr) {
        switch (ch) {
            // Jika karakter adalah tanda kurung buka, masukkan ke dalam
            tumpukan
            case '(':
            case '[':
            case '{':

```

```

        stack.push(ch);
        break;
    // Jika karakter adalah tanda kurung tutup
    case ')':
        // Jika tumpukan kosong atau tanda kurung teratas dalam
tumpukan bukan tanda kurung buka yang sesuai
        if (stack.empty() || stack.top() != '(') {
            return false; // Ekspresi tidak seimbang
        }
        stack.pop(); // Hapus tanda kurung buka yang sesuai dari
tumpukan
        break;
    case ']':
        if (stack.empty() || stack.top() != '[') {
            return false;
        }
        stack.pop();
        break;
    case '}':
        if (stack.empty() || stack.top() != '{') {
            return false;
        }
        stack.pop();
        break;
    default:
        break;
}

// Jika tumpukan kosong, berarti semua tanda kurung sudah seimbang
return stack.empty();
}

// Program utama
int main() {
    string expr1 = "{()}"; // Ekspresi pertama
    string expr2 = "{()}}"; // Ekspresi kedua

    // Memeriksa dan menampilkan apakah ekspresi pertama seimbang atau
tidak
    cout << "Expression: " << expr1 << " is "
        << (isBalancedParenthesis(expr1) ? "balanced" : "not
balanced") << endl;
    // Memeriksa dan menampilkan apakah ekspresi kedua seimbang atau
tidak
    cout << "Expression: " << expr2 << " is "

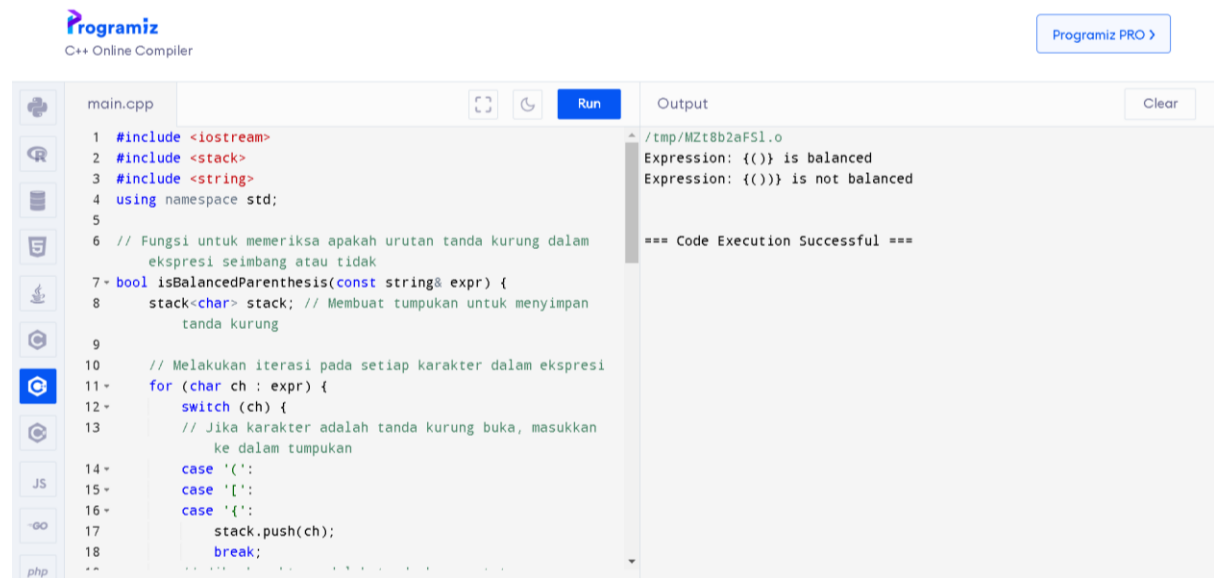
```

```

        << (isBalancedParenthesis(expr2) ? "balanced" : "not
balanced") << endl;

    return 0;
}

```



The screenshot shows the Programiz C++ Online Compiler interface. The code editor on the left contains a C++ program named `main.cpp` that defines a function `isBalancedParenthesis` to check if a string of parentheses is balanced. The program uses a stack to store opening parentheses and iterates through the string, pushing opening parentheses and popping them when a closing parenthesis is encountered. The output window on the right shows the results of two test cases: `Expression: {} is balanced` and `Expression: {}{} is not balanced`, followed by a success message: `=== Code Execution Successful ===`.

```

1 #include <iostream>
2 #include <stack>
3 #include <string>
4 using namespace std;
5
6 // Fungsi untuk memeriksa apakah urutan tanda kurung dalam
  ekspresi seimbang atau tidak
7 bool isBalancedParenthesis(const string& expr) {
8     stack<char> stack; // Membuat tumpukan untuk menyimpan
      tanda kurung
9
10    // Melakukan iterasi pada setiap karakter dalam ekspresi
11    for (char ch : expr) {
12        switch (ch) {
13            // Jika karakter adalah tanda kurung buka, masukkan
             ke dalam tumpukan
14            case '(':
15            case '[':
16            case '{':
17                stack.push(ch);
18                break;

```

### 3. Konversi Infix ke Postfix

Mengonversi ekspresi matematis dari notasi infix ke postfix menggunakan Stack untuk menangani prioritas operator sesuai dengan aturan precedence.

```

#include <iostream>
#include <stack>
#include <string>
#include <cctype>
using namespace std;

// Fungsi untuk memeriksa apakah karakter adalah operator
bool isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Fungsi untuk menentukan tingkat prioritas operator
int precedence(char op) {
    if (op == '+' || op == '-') {
        return 1;
    }
    else if (op == '*' || op == '/') {

```

```

        return 2;
    }
    return 0;
}

// Fungsi untuk mengonversi ekspresi infix menjadi postfix
string infixToPostfix(const string& infix) {
    stack<char> stack; // Membuat tumpukan untuk menyimpan operator
    string postfix; // String untuk menyimpan ekspresi postfix

    // Melakukan iterasi pada setiap karakter dalam ekspresi infix
    for (char ch : infix) {
        // Jika karakter adalah huruf atau angka, tambahkan langsung
        ke ekspresi postfix
        if (isalnum(ch)) {
            postfix += ch;
        }
        // Jika karakter adalah tanda kurung buka, masukkan ke dalam
        tumpukan
        else if (ch == '(') {
            stack.push(ch);
        }
        // Jika karakter adalah tanda kurung tutup
        else if (ch == ')') {
            // Keluarkan operator dari tumpukan dan tambahkan ke
            ekspresi postfix
            while (!stack.empty() && stack.top() != '(') {
                postfix += stack.top();
                stack.pop();
            }
            // Hapus tanda kurung buka dari tumpukan jika ada
            if (!stack.empty() && stack.top() == '(') {
                stack.pop();
            }
        }
        // Jika karakter adalah operator
        else {
            // Keluarkan operator dari tumpukan yang memiliki
            prioritas yang lebih tinggi atau sama
            while (!stack.empty() && precedence(ch) <=
precedence(stack.top())) {
                postfix += stack.top();
                stack.pop();
            }
            // Masukkan operator ke dalam tumpukan
            stack.push(ch);
        }
    }
}

```

```

    }

    // Keluarkan semua operator dari tumpukan dan tambahkan ke
    ekspresi postfix
    while (!stack.empty()) {
        postfix += stack.top();
        stack.pop();
    }

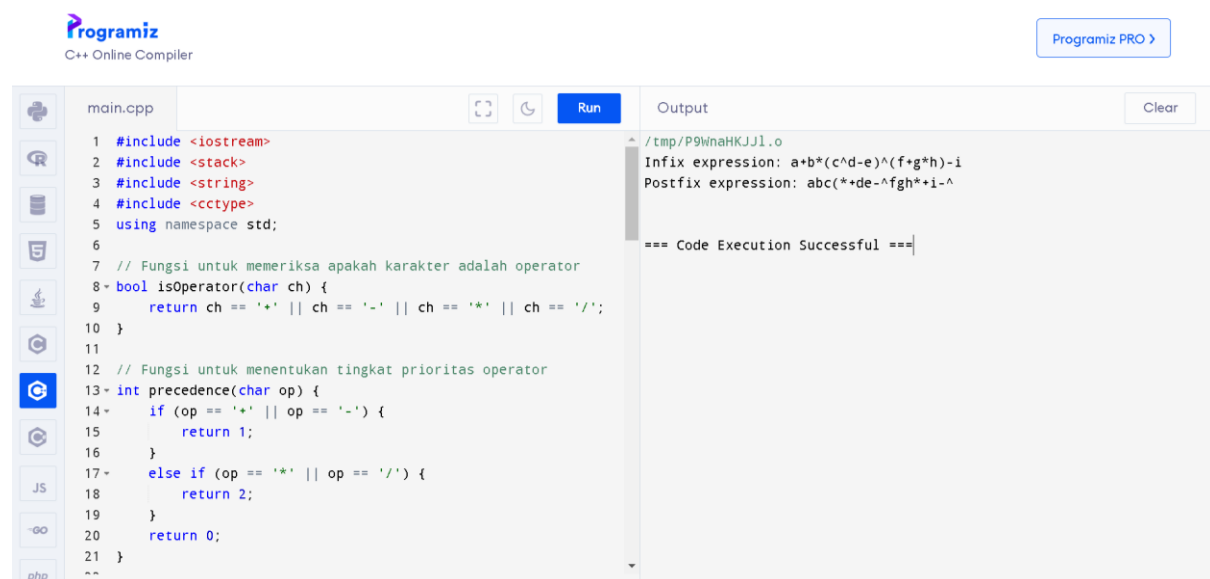
    // Kembalikan ekspresi postfix yang telah dihasilkan
    return postfix;
}

// Program utama
int main() {
    string infix = "a+b*(c^d-e)^(f+g*h)-i"; // Ekspresi infix yang
    akan diubah menjadi postfix
    string postfix = infixToPostfix(infix); // Konversi ekspresi infix
    menjadi postfix

    // Tampilkan ekspresi infix dan postfix yang dihasilkan
    cout << "Infix expression: " << infix << endl;
    cout << "Postfix expression: " << postfix << endl;

    return 0;
}

```



The screenshot shows the Programiz C++ Online Compiler interface. On the left, there's a sidebar with icons for various programming languages. The main area displays the C++ code for infix to postfix conversion. The code includes headers for `<iostream>`, `<stack>`, `<string>`, and `<cctype>`, and uses the `std` namespace. It defines two helper functions: `isOperator` to check if a character is an operator (+, -, \*, /) and `precedence` to determine the priority of operators. The main function initializes the infix expression "a+b\*(c^d-e)^(f+g\*h)-i" and calls `infixToPostfix` to convert it to postfix. The output window on the right shows the infix and postfix expressions and a success message.

```

main.cpp
1 #include <iostream>
2 #include <stack>
3 #include <string>
4 #include <cctype>
5 using namespace std;
6
7 // Fungsi untuk memeriksa apakah karakter adalah operator
8 bool isOperator(char ch) {
9     return ch == '+' || ch == '-' || ch == '*' || ch == '/';
10 }
11
12 // Fungsi untuk menentukan tingkat prioritas operator
13 int precedence(char op) {
14     if (op == '+' || op == '-') {
15         return 1;
16     }
17     else if (op == '*' || op == '/') {
18         return 2;
19     }
20     return 0;
21 }

```

Output

```

/tmp/P9WnaHKJl.o
Infix expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression: abc(*+de-^fgh*+i-^

=== Code Execution Successful ===

```

## 4. Konversi Infix ke Prefix



Mengonversi ekspresi matematis dari notasi infix ke prefix dengan membalik ekspresi infix, mengonversinya ke postfix, lalu membalik kembali hasilnya.

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>
#include <cctype>
using namespace std;

// Fungsi untuk menentukan tingkat prioritas operator
int precedence(char op) {
    if (op == '+' || op == '-') {
        return 1;
    }
    else if (op == '*' || op == '/') {
        return 2;
    }
    else if (op == '^') {
        return 3;
    }
    return 0;
}

// Fungsi untuk mengonversi ekspresi infix menjadi prefix
string infixToPrefix(const string& infix) {
    string reversedInfix = infix;
    reverse(reversedInfix.begin(), reversedInfix.end()); //
    Membalikkan ekspresi infix

    // Mengubah tanda kurung menjadi tanda kurung sebaliknya
    for (int i = 0; i < reversedInfix.length(); i++) {
        if (reversedInfix[i] == '(') {
            reversedInfix[i] = ')';
        }
        else if (reversedInfix[i] == ')') {
            reversedInfix[i] = '(';
        }
    }

    stack<char> stack; // Membuat tumpukan untuk menyimpan operator
    string prefix; // String untuk menyimpan ekspresi prefix

    // Melakukan iterasi pada setiap karakter dalam ekspresi infix
    yang telah dibalik
    for (char ch : reversedInfix) {
```

```

        // Jika karakter adalah huruf atau angka, tambahkan langsung
ke ekspresi prefix
        if (isalnum(ch)) {
            prefix += ch;
        }
        // Jika karakter adalah tanda kurung tutup
        else if (ch == ')') {
            stack.push(ch); // Masukkan ke dalam tumpukan
        }
        // Jika karakter adalah tanda kurung buka
        else if (ch == '(') {
            // Keluarkan operator dari tumpukan dan tambahkan ke
ekspresi prefix
            while (!stack.empty() && stack.top() != ')') {
                prefix += stack.top();
                stack.pop();
            }
            // Hapus tanda kurung tutup dari tumpukan jika ada
            if (!stack.empty() && stack.top() == ')') {
                stack.pop();
            }
        }
        // Jika karakter adalah operator
        else {
            // Keluarkan operator dari tumpukan yang memiliki
prioritas yang lebih tinggi atau sama
            while (!stack.empty() && stack.top() != ')' &&
                ((ch != '^' && precedence(ch) <=
precedence(stack.top())) ||
                (ch == '^' && precedence(ch) <
precedence(stack.top())))) {
                prefix += stack.top();
                stack.pop();
            }
            // Masukkan operator ke dalam tumpukan
            stack.push(ch);
        }
    }

    // Keluarkan semua operator dari tumpukan dan tambahkan ke
ekspresi prefix
    while (!stack.empty()) {
        prefix += stack.top();
        stack.pop();
    }

```

```

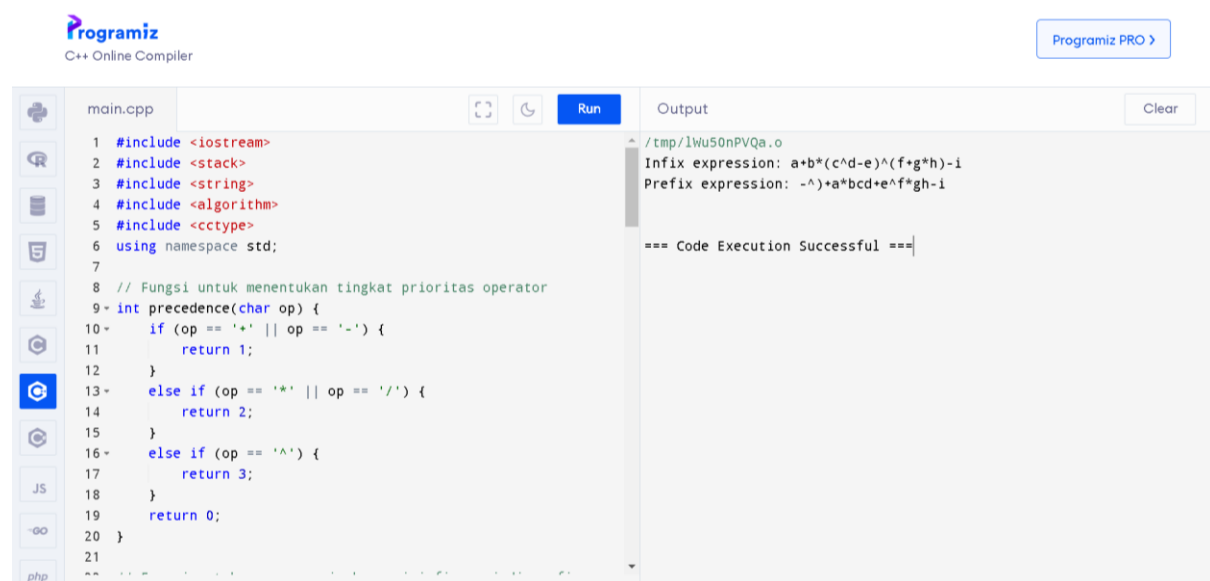
        reverse(prefix.begin(), prefix.end()); // Membalikkan ekspresi
        prefix kembali ke posisi semula
        return prefix; // Kembalikan ekspresi prefix yang telah dihasilkan
    }

// Program utama
int main() {
    string infix = "a+b*(c^d-e)^(f+g*h)-i"; // Ekspresi infix yang
    akan diubah menjadi prefix
    string prefix = infixToPrefix(infix); // Konversi ekspresi infix
    menjadi prefix

    // Tampilkan ekspresi infix dan prefix yang dihasilkan
    cout << "Infix expression: " << infix << endl;
    cout << "Prefix expression: " << prefix << endl;

    return 0;
}

```



The screenshot shows the Programiz C++ Online Compiler interface. On the left, the code editor displays the C++ program for infix to prefix conversion. The code includes headers for `<iostream>`, `<stack>`, `<string>`, `<algorithm>`, and `<cctype>`, and uses the `std` namespace. It defines a `precedence` function to determine operator priority and a `infixToPrefix` function (partially visible). The `main` function initializes the infix expression `"a+b*(c^d-e)^(f+g*h)-i"` and calls `infixToPrefix` to convert it to prefix notation. On the right, the Output panel shows the program's execution results: the infix expression and its corresponding prefix expression `-(^)+a*bcd+e^f*gh-i`. A message at the bottom of the output panel states "=== Code Execution Successful ===".

## 5. Evaluasi Ekspresi Postfix

Mengevaluasi ekspresi postfix dengan mendorong operand ke dalam Stack dan melakukan operasi aritmatika sesuai operator yang ditemukan

```

#include <iostream>
#include <stack>
#include <string>
#include <cctype>
using namespace std;

```

```

// Fungsi untuk mengevaluasi ekspresi postfix
int evaluatePostfix(const string& postfix) {
    stack<int> stack; // Membuat tumpukan untuk menampung operand

    // Melakukan iterasi pada setiap karakter dalam ekspresi postfix
    for (char ch : postfix) {
        // Jika karakter merupakan angka, masukkan ke dalam tumpukan
        if (isdigit(ch)) {
            stack.push(ch - '0');
        }
        // Jika karakter merupakan operator
        else {
            // Ambil dua operand teratas dari tumpukan
            int operand2 = stack.top();
            stack.pop();
            int operand1 = stack.top();
            stack.pop();

            // Lakukan operasi sesuai dengan operator
            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;
            }
        }
    }

    // Kembalikan hasil akhir dari evaluasi ekspresi postfix
    return stack.top();
}

// Program utama
int main() {
    string postfix = "83+72*-"; // Ekspresi postfix yang akan
    dievaluasi
    int result = evaluatePostfix(postfix); // Evaluasi ekspresi
    postfix

```

```

// Tampilkan ekspresi postfix dan hasil evaluasinya
cout << "Postfix expression: " << postfix << endl;
cout << "Result: " << result << endl;

return 0;
}

```

The screenshot shows the Programiz C++ Online Compiler interface. The code in main.cpp defines a function `evaluatePostfix` that takes a string postfix expression and returns its integer result. It uses a `stack<int>` to store operands. The example expression is `83*72*-`, which evaluates to `-3`. The output window shows the execution result: `Postfix expression: 83*72*-` and `Result: -3`, followed by a success message.

## 6. Palindrome String menggunakan Stack

Untuk memeriksa apakah sebuah string merupakan palindrome atau tidak dengan membandingkan setengah karakter pertama yang didorong ke dalam Stack dengan setengah karakter kedua

```

#include <iostream>
#include <stack>
#include <string>
#include <algorithm>
using namespace std;

```

```

// Fungsi untuk memeriksa apakah sebuah string merupakan palindrome
bool isPalindrome(const string& str) {
    stack<char> stack; // Membuat tumpukan untuk menyimpan karakter
    pertama setengah dari string
    string temp = str; // Salin string ke variabel sementara

```

```

    // Menghapus spasi dan mengubah semua huruf menjadi huruf kecil
    temp.erase(remove_if(temp.begin(), temp.end(), ::isspace),
temp.end());
    transform(temp.begin(), temp.end(), temp.begin(), ::tolower);

```

```

// Mendorong setengah karakter pertama ke dalam tumpukan

```

```

    for (int i = 0; i < temp.length() / 2; i++) {
        stack.push(temp[i]);
    }

    // Mulai dari tengah (atau tengah + 1 jika panjang string ganjil)
    int start = temp.length() % 2 == 0 ? temp.length() / 2 :
(temp.length() / 2) + 1;

    // Memeriksa karakter yang sesuai dengan tumpukan
    for (int i = start; i < temp.length(); i++) {
        if (temp[i] != stack.top()) {
            return false; // Tidak merupakan palindrome
        }
        stack.pop();
    }

    return true; // String merupakan palindrome
}

// Fungsi utama program
int main() {
    string str; // Variabel untuk menyimpan string dari pengguna
    cout << "Masukkan sebuah string: ";
    getline(cin, str); // Membaca string dari input pengguna

    // Memeriksa apakah string merupakan palindrome dan menampilkan
    hasilnya
    cout << "\"" << str << "\" is " << (isPalindrome(str) ? "a
palindrome" : "not a palindrome") << endl;

    return 0;
}

```

The screenshot shows the Programiz C++ Online Compiler interface. On the left, there's a sidebar with icons for various programming languages. The main area displays a C++ program that checks if a string is a palindrome. The code includes headers for `<iostream>`, `<stack>`, `<string>`, and `<algorithm>`, and uses the `std` namespace. The `isPalindrome` function uses a stack to compare the first half of the string with the second half. The `main` function prompts the user to enter a string, reads it, and then calls `isPalindrome` to check if it's a palindrome. The output shows that the input string "malam" is indeed a palindrome.

```

main.cpp
1 #include <iostream>
2 #include <stack>
3 #include <string>
4 #include <algorithm>
5 using namespace std;
6
7 // Fungsi untuk memeriksa apakah sebuah string merupakan
  palindrome
8 bool isPalindrome(const string& str) {
9     stack<char> stack; // Membuat tumpukan untuk menyimpan
    karakter pertama setengah dari string
10    string temp = str; // Salin string ke variabel sementara
11
12    // Menghapus spasi dan mengubah semua huruf menjadi huruf
    kecil
13    temp.erase(remove_if(temp.begin(), temp.end(), ::isspace
    ), temp.end());
14    transform(temp.begin(), temp.end(), temp.begin(),
    ::tolower);
15
16    // Mendorong setengah karakter pertama ke dalam tumpukan

```

Output

```

/tmp/CfspZC5AZO.o
Masukkan sebuah string: malam
"malam" is a palindrome

=== Code Execution Successful ===

```