# ALGORITMA DAN STUKTUR DATA

## Stack


**Oleh :**

Fina Salsabila Pramudita (5223600006)

**Program Studi Sarjana Terapan Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika Negeri Surabaya**


**2024**

1. Implementasi stack menggunakan linked list

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

class Node {
public:
    int data; // Mengganti variabel 'value' menjadi 'data' untuk
lebih jelas
    Node* nextNode; // Mengganti variabel 'next' menjadi 'nextNode'
untuk lebih jelas

    // Konstruktor untuk membuat simpul baru dengan nilai tertentu
dan pointer next yang menunjuk ke nullptr
    Node(int val, Node* n = nullptr) {
        data = val;
        nextNode = n;
    }
};

class ListStack {
private:
    Node* top; // Mengganti variabel 'head' menjadi 'top' untuk
lebih jelas
    int itemCount; // Mengganti variabel 'count' menjadi
'itemCount' untuk lebih jelas

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

    // Mengembalikan jumlah elemen dalam tumpukan
    int getSize() {
        return itemCount;
```

```
    }

    // Mengembalikan true jika tumpukan kosong, false jika tidak
    bool isEmpty() {
        return itemCount == 0;
    }

    // Mengembalikan nilai dari elemen teratas tumpukan tanpa menghapusnya
    int peek() {
        if (isEmpty()) {
            throw out_of_range("Stack is empty");
        }
        return top->data;
    }

    // Menambahkan elemen baru ke tumpukan
    void push(int val) {
        top = new Node(val, top);
        itemCount++;
    }

    // Menghapus dan mengembalikan nilai dari elemen teratas tumpukan
    int pop() {
        if (isEmpty()) {
            throw out_of_range("Stack is empty");
        }
        int val = top->data;
        Node* temp = top;
        top = top->nextNode;
        delete temp;
        itemCount--;
        return val;
    }
};

int main() {
    ListStack stack;
```

```
        stack.push(1);
        stack.push(2);
        stack.push(3);

        cout << "Elemen paling atas: " << stack.peek() << endl;
        cout << "Ukuran stack: " << stack.getSize() << endl;

        stack.pop();
        cout << "Ukuran teratas setelah pop: " << stack.peek() << endl;

        return 0;
    }
```

- Class node Mendefinisikan sebuah simpul (node) dalam linked list dengan atribut data (nilai) dan pointer ke node berikutnya.
- Kelas ListStack : Mendefinisikan tumpukan menggunakan linked list. Kelas ini memiliki atribut `top` (pointer ke elemen teratas tumpukan) dan `itemCount` (jumlah elemen dalam tumpukan).
- Metode-metode ListStack**:
  - `getSize()`: Mengembalikan jumlah elemen dalam tumpukan.
  - `isEmpty()`: Mengembalikan true jika tumpukan kosong, dan false jika tidak.
  - `peek()`: Mengembalikan nilai dari elemen teratas tumpukan tanpa menghapusnya.
  - `push(int val)`: Menambahkan elemen baru ke tumpukan.
  - `pop()`: Menghapus dan mengembalikan nilai dari elemen teratas tumpukan.
- Fungsi `main()`: Pengujian dari implementasi tumpukan. Beberapa elemen ditambahkan ke tumpukan, dan kemudian nilai dari elemen teratas dan ukuran tumpukan dicetak. Salah satu elemen kemudian dihapus dari tumpukan, dan nilai elemen teratas setelah penghapusan dicetak lagi.

2. Memeriksa keseimbangan tanda kurung

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

    // Fungsi untuk mengecek apakah urutan tanda kurung dalam
    ekspresi diberikan seimbang atau tidak
    bool isKurungSeimbang(const string& ekspresi) {
        stack<char> tumpukanKurung; // Membuat tumpukan untuk melacak
    tanda kurung yang belum ditutup

        // Melintasi setiap karakter dalam ekspresi
        for (char ch : ekspresi) {
            switch (ch) {
                case '(':
                case '[':
                case '{':
```

```
                tumpukanKurung.push(ch); // Menambahkan tanda
kurung pembuka ke tumpukan
                break;
            case ')':
                // Jika tumpukan kosong atau tanda kurung teratas
di tumpukan tidak sesuai dengan tanda kurung buka yang sesuai,
                // ekspresi tidak seimbang
                if (tumpukanKurung.empty() ||
tumpukanKurung.top() != '(') {
                    return false;
                }
                tumpukanKurung.pop(); // Menghapus tanda kurung
buka yang sesuai dari tumpukan
                break;
            case ']':
                if (tumpukanKurung.empty() ||
tumpukanKurung.top() != '[') {
                    return false;
                }
                tumpukanKurung.pop();
                break;
            case '}':
                if (tumpukanKurung.empty() ||
tumpukanKurung.top() != '{') {
                    return false;
                }
                tumpukanKurung.pop();
                break;
            default:
                break;
        }
    }

    // Jika tumpukan kosong pada akhir iterasi, itu berarti semua
tanda kurung telah seimbang
    return tumpukanKurung.empty();
}

int main() {
    string ekspresi1 = "{()}";
    string ekspresi2 = "{())}";

    // Menguji fungsi isKurungSeimbang pada dua ekspresi berbeda
dan mencetak hasilnya
    cout << "Ekspresi: " << ekspresi1 << " adalah "
         << (isKurungSeimbang(ekspresi1) ? "seimbang" : "tidak
seimbang") << endl;
    cout << "Ekspresi: " << ekspresi2 << " adalah "
         << (isKurungSeimbang(ekspresi2) ? "seimbang" : "tidak
seimbang") << endl;

    return 0;
```

```
}
```

Skrip ini mengevaluasi apakah urutan tanda kurung dalam ekspresi diberikan seimbang atau tidak. Ini dilakukan dengan menggunakan stack untuk melacak tanda kurung yang dibuka dan memastika bahwa setiap tanda kurung ditutup sesuai dengan tanda kurung yang dibuka terakhir.

- Fungsi isKurungSeimbang:
  - Menerima sebuah string ekspresi yang berisi urutan karakter tanda kurung.
  - Menggunakan tumpukan (stack<char>) untuk melacak tanda kurung yang belum ditutup.
  - Melintasi setiap karakter dalam ekspresi:
  - Jika karakter tersebut adalah tanda kurung pembuka, itu ditambahkan ke tumpukan.
  - Jika karakter tersebut adalah tanda kurung penutup, itu memeriksa apakah tumpukan kosong atau apakah tanda kurung teratas di tumpukan sesuai dengan tanda kurung pembuka yang cocok. Jika tidak, itu mengembalikan false.
  - Jika semua tanda kurung sudah seimbang dan pada akhirnya tumpukan kosong, itu mengembalikan true. Jika tidak, itu mengembalikan false.
  - Menguji fungsi isKurungSeimbang pada dua ekspresi berbeda (expr1 dan expr2).
  - Mencetak hasilnya ("seimbang" jika ekspresi seimbang dan "tidak seimbang" jika tidak).

3. Konversi Infix ke postfix

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

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

// Fungsi untuk mengembalikan prioritas operator matematika
int prioritas(char op) {
    if (op == '+' || op == '-') {
        return 1;
    } else if (op == '*' || op == '/') {
        return 2;
    }
    return 0;
}

// Fungsi untuk mengubah ekspresi infix menjadi postfix
string infixKePostfix(const string& infix) {
    stack<char> tumpukan;
    string postfix;

    for (char ch : infix) {
        if (isalnum(ch)) { // Jika karakter adalah operand (angka
atau huruf)
```

```
            postfix += ch; // Tambahkan langsung ke ekspresi postfix
        } else if (ch == '(') { // Jika karakter adalah kurung buka
            tumpukan.push(ch); // Masukkan ke tumpukan
        } else if (ch == ')') { // Jika karakter adalah kurung tutup
            // Pop dan tambahkan operator-operator yang ada di tumpukan hingga menemui kurung buka yang sesuai
            while (!tumpukan.empty() && tumpukan.top() != '(') {
                postfix += tumpukan.top();
                tumpukan.pop();
            }
            // Hapus kurung buka dari tumpukan (jika ada)
            if (!tumpukan.empty() && tumpukan.top() == '(') {
                tumpukan.pop();
            }
        } else { // Jika karakter adalah operator
            // Pop dan tambahkan operator-operator yang memiliki prioritas lebih tinggi atau sama dengan operator saat ini
            while (!tumpukan.empty() && prioritas(ch) <= prioritas(tumpukan.top())) {
                postfix += tumpukan.top();
                tumpukan.pop();
            }
            // Masukkan operator saat ini ke tumpukan
            tumpukan.push(ch);
        }
    }

    // Pop dan tambahkan sisa operator-operator yang tersisa di tumpukan
    while (!tumpukan.empty()) {
        postfix += tumpukan.top();
        tumpukan.pop();
    }

    return postfix;
}

int main() {
    string infix = "a+b*(c^d-e)^(f+g*h)-i";
    string postfix = infixKePostfix(infix);

    cout << "Ekspresi infix: " << infix << endl;
    cout << "Ekspresi postfix: " << postfix << endl;

    return 0;
}
```

- Fungsi `adalahOperator`:
  - Memeriksa apakah sebuah karakter merupakan operator matematika.
- Fungsi `prioritas`:

- Mengembalikan prioritas operator matematika.

- Fungsi `infixKePostfix`:
  - Menerima ekspresi matematika dalam bentuk infix dan mengonversinya menjadi bentuk postfix.
  - Menggunakan tumpukan untuk menyimpan operator dan memastikan prioritas operasi yang tepat.
- Fungsi `main`:
  - Membuat ekspresi matematika infix.
  - Memanggil fungsi `infixKePostfix` untuk mengonversi infix menjadi postfix.
  - Mencetak hasil ekspresi infix dan postfix.

4. Konversi infix ke prefix

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

// Fungsi untuk menentukan prioritas operator
int prioritas(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 infixKePrefix(const string& infix) {
    // Membalik ekspresi infix
    string infixTerbalik = infix;
    reverse(infixTerbalik.begin(), infixTerbalik.end());

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

    stack<char> tumpukan;
    string prefix;
```

```
    // Melintasi ekspresi infix terbalik
    for (char ch : infixTerbalik) {
        if (isalnum(ch)) { // Jika karakter adalah operand (angka
atau huruf)
            prefix += ch; // Tambahkan langsung ke ekspresi prefix
        } else if (ch == ')') { // Jika karakter adalah kurung
tutup
            tumpukan.push(ch); // Masukkan ke tumpukan
        } else if (ch == '(') { // Jika karakter adalah kurung buka
            // Pop dan tambahkan operator-operator yang ada di
tumpukan hingga menemui kurung tutup yang sesuai
            while (!tumpukan.empty() && tumpukan.top() != ')') {
                prefix += tumpukan.top();
                tumpukan.pop();
            }
            // Hapus kurung tutup dari tumpukan (jika ada)
            if (!tumpukan.empty() && tumpukan.top() == ')') {
                tumpukan.pop();
            }
        } else { // Jika karakter adalah operator
            // Pop dan tambahkan operator-operator yang memiliki
prioritas lebih tinggi atau sama dengan operator saat ini
            while (!tumpukan.empty() && tumpukan.top() != ')' &&
                   ((ch    !=    '^'    &&    prioritas(ch)    <=
prioritas(tumpukan.top())) ||
                    (ch    ==    '^'    &&    prioritas(ch)    <
prioritas(tumpukan.top())))) {
                prefix += tumpukan.top();
                tumpukan.pop();
            }
            // Masukkan operator saat ini ke tumpukan
            tumpukan.push(ch);
        }
    }

    // Pop dan tambahkan sisa operator-operator yang tersisa di
tumpukan
    while (!tumpukan.empty()) {
        prefix += tumpukan.top();
        tumpukan.pop();
    }

    //  Balik  kembali  ekspresi  prefix  karena  ekspresi  awalnya
dibalik
    reverse(prefix.begin(), prefix.end());
    return prefix;
}

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

```
        string prefix = infixKePrefix(infix);

        // Cetak hasil ekspresi infix dan prefix
        cout << "Ekspresi infix: " << infix << endl;
        cout << "Ekspresi prefix: " << prefix << endl;

        return 0;
    }
```

- Fungsi prioritas:
  - Menentukan prioritas operator.
- Fungsi infixKePrefix:
  - Mengonversi ekspresi matematika dari infix menjadi prefix.
  - Memanipulasi ekspresi infix dengan membaliknya dan mengubah tanda kurung.
  - Menggunakan tumpukan untuk menyimpan operator dan memastikan urutan operasi yang benar.
  - Menghasilkan ekspresi prefix yang sesuai.
- Fungsi main:
  - Mendefinisikan ekspresi matematika infix.
  - Memanggil fungsi infixKePrefix untuk mengonversi ekspresi tersebut menjadi prefix.
  - Mencetak hasil ekspresi infix dan prefix.

5. Evaluasi ekspresi portfix

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

    // Fungsi untuk mengevaluasi ekspresi postfix
    int evaluasiPostfix(const string& postfix) {
        stack<int> tumpukan;

        // Melintasi setiap karakter dalam ekspresi postfix
        for (char ch : postfix) {
            if (isdigit(ch)) { // Jika karakter adalah angka
                tumpukan.push(ch - '0'); // Masukkan angka ke dalam
    tumpukan
            } else { // Jika karakter adalah operator
                // Pop dua operand teratas dari tumpukan
                int operand2 = tumpukan.top();
                tumpukan.pop();
                int operand1 = tumpukan.top();
                tumpukan.pop();

                // Evaluasi operasi dan hasilnya dimasukkan kembali ke
    dalam tumpukan
                switch (ch) {
                    case '+':
                        tumpukan.push(operand1 + operand2);
                        break;
```

```
                case '-':
                    tumpukan.push(operand1 - operand2);
                    break;
                case '*':
                    tumpukan.push(operand1 * operand2);
                    break;
                case '/':
                    tumpukan.push(operand1 / operand2);
                    break;
            }
        }
    }

    // Hasil akhir evaluasi adalah nilai yang tersisa di tumpukan
    return tumpukan.top();
}

int main() {
    // Ekspresi postfix
    string postfix = "83+72*-";
    // Evaluasi ekspresi postfix
    int hasil = evaluasiPostfix(postfix);

    // Cetak ekspresi postfix dan hasil evaluasinya
    cout << "Ekspresi postfix: " << postfix << endl;
    cout << "Hasil: " << hasil << endl;

    return 0;
}
```

- Fungsi evaluasiPostfix:
  - Menerima ekspresi matematika dalam bentuk postfix dan mengembalikan hasil evaluasinya.
  - Menggunakan tumpukan untuk memproses operand dan operator.
  - Melintasi setiap karakter dalam ekspresi postfix, memproses angka sebagai operand dan operator sebagai operasi matematika.
  - Hasil akhir evaluasi adalah nilai yang tersisa di tumpukan setelah semua operasi selesai dievaluasi.
- Fungsi main:
  - Mendefinisikan ekspresi matematika postfix.
  - Memanggil fungsi evaluasiPostfix untuk mengevaluasi ekspresi tersebut.
  - Mencetak ekspresi postfix dan hasil evaluasinya.