ALGORITMA DAN STUKTUR DATA

Queue

Oleh :

Fina Salsabila Pramudita (5223600006)

Program Studi Sarjana Terapan Teknologi Game

Departemen Teknologi Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2024

1. Queue menggunakan linked list

   Dalam implementasi queue menggunakan linked list, kita menggunakan struktur data linked list untuk menyimpan elemen – elemen queue.

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

class Queue {
private:
    // Struktur node untuk merepresentasikan elemen dalam antrian
    struct Node {
        int value;  // Nilai dari node
        Node* next; // Pointer ke node berikutnya dalam antrian
        Node(int v, Node* n) : value(v), next(n) {} // Konstruktor
node
    };

    Node* head = nullptr; // Pointer ke kepala antrian
    Node* tail = nullptr; // Pointer ke ekor antrian
    int count = 0; // Jumlah elemen dalam antrian

public:
    // Fungsi untuk mendapatkan jumlah elemen dalam antrian
    int size() { return count; }

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

    // Fungsi untuk mencetak elemen dalam antrian
    void print() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->value << " "; // Mencetak nilai node
            temp = temp->next; // Pindah ke node berikutnya
        }
        cout << endl;
    }

    // Fungsi untuk melihat nilai elemen yang paling depan dalam
antrian
```

```
    int peek() {
        if (empty()) {
            throw invalid_argument("QueueEmptyException");
        }
        return head->value; // Mengembalikan nilai dari kepala antrian
    }

    // Fungsi untuk menambahkan elemen baru ke dalam antrian
    void enqueue(int value) {
        Node* temp = new Node(value, nullptr); // Membuat node baru
        if (head == nullptr) {
            head = tail = temp; // Jika antrian kosong, node baru adalah kepala dan ekor
        } else {
            tail->next = temp; // Mengaitkan node baru ke ekor antrian
            tail = temp; // Menggeser ekor ke node baru
        }
        count++; // Menambah jumlah elemen dalam antrian
    }

    // Fungsi untuk menghapus elemen dari antrian
    int dequeue() {
        if (empty()) {
            throw invalid_argument("QueueEmptyException");
        }
        int value = head->value; // Menyimpan nilai dari kepala antrian yang akan dihapus
        Node* temp = head; // Menyimpan pointer ke kepala antrian
        head = head->next; // Menggeser kepala antrian ke node berikutnya
        delete temp; // Menghapus node yang sudah tidak diperlukan
        count--; // Mengurangi jumlah elemen dalam antrian
        return value; // Mengembalikan nilai dari node yang dihapus
    }
};

int main() {
```

```
        Queue q; // Membuat objek antrian
        // Menambahkan nilai dari 1 hingga 100 ke dalam antrian
        for (int i = 1; i <= 100; i++) {
            q.enqueue(i);
        }
        // Menghapus 50 elemen pertama dari antrian
        for (int i = 1; i <= 50; i++) {
            q.dequeue();
        }
        q.print(); // Mencetak elemen yang tersisa dalam antrian
        return 0;
    }
```

2. Enqueue pada queue linked list

   Program ini adalah implementasi untuk menambahkan elemenbaru di akhir dari struktur data queue menggunakan linked list.

```
       void Queue::enqueue(int value) {
           Node* newNode = new Node(value, nullptr);
           if (head == nullptr) {
               head = tail = newNode;
           } else {
               tail->next = newNode;
               tail = newNode;
           }
           count++;
}
```

3. Dequeue pada queue linked list

   Digunakan untuk menghapus elemen dari awal queue linked list

```
       int Queue::dequeue() {
           if (empty()) {
               throw invalid_argument("QueueEmptyException");
           }
           int val = head->value;
           Node* temp_node = head;
           head = head->next;
           delete temp_node;
           count--;
           return val;
}
```

4. Implementtasi queue menggunakan 2 stack

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

class QueueUsingStack {
private:
    stack<int> inputStack;  // stack input untuk operasi enqueue
    stack<int> outputStack; // stack output untuk operasi dequeue

public:
    // Fungsi untuk menambahkan elemen ke dalam antrian
    void enqueue(int value) {
        inputStack.push(value); // Masukkan elemen ke dalam stack
input
    }

    //  Fungsi  untuk  menghapus  elemen  dari  depan  antrian  dan
mengembalikan nilainya
    int dequeue() {
        // Jika stack output kosong, pindahkan semua elemen dari
stack input ke stack output
        if (outputStack.empty()) {
            while (!inputStack.empty()) {
                int value = inputStack.top(); // Ambil nilai dari
elemen teratas stack input
                inputStack.pop();  //  Hapus  elemen  teratas  dari
stack input
                outputStack.push(value); // Masukkan nilai ke dalam
stack output
            }
        }

        // Jika setelah proses pindah stack input ke stack output
stack output masih kosong, kembalikan exception
        if (outputStack.empty()) {
            throw invalid_argument("QueueEmptyException");
        }
```

```
        // Ambil nilai dari elemen teratas stack output untuk
dequeue
        int value = outputStack.top();
        outputStack.pop(); // Hapus elemen teratas dari stack
output
        return value; // Kembalikan nilai elemen yang di-dequeue
    }
};

int main() {
    QueueUsingStack queue;
    queue.enqueue(1);
    queue.enqueue(11);
    queue.enqueue(111);
    cout << queue.dequeue() << endl; // Output: 1
    queue.enqueue(2);
    queue.enqueue(21);
    queue.enqueue(211);
    cout << queue.dequeue() << endl; // Output: 11
    cout << queue.dequeue() << endl; // Output: 111
    return 0;
}
```

TUGAS

1. Reverse a Stack

   Memindahkan semua elemen dari tumpukan ke antrian. Kemudian memindahkan semua elemen dari antrian Kembali ke tumpukan. Reserve a stack ini digunakan untuk membalikan isi dari stack yang diberikan.

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

    // Fungsi untuk membalikkan stack
    void reverseStack(stack<int>& stk) {
        queue<int> que; // Queue untuk menyimpan elemen-elemen
    stack

        // Step 1: Memindahkan semua elemen dari stack ke queue
        while (!stk.empty()) {
            que.push(stk.top()); // Memasukkan elemen dari stack
    ke queue
            stk.pop(); // Menghapus elemen yang telah dipindahkan
    dari stack
```

```
        }

        // Step 2: Memindahkan semua elemen dari queue kembali ke
    stack
        while (!que.empty()) {
            stk.push(que.front()); // Memasukkan elemen dari queue
    kembali ke stack
            que.pop(); // Menghapus elemen yang telah dipindahkan
    dari queue
        }
    }

    int main() {
        stack<int> originalStack;

        // Menambahkan elemen ke dalam stack
        originalStack.push(1);
        originalStack.push(2);
        originalStack.push(3);
        originalStack.push(4);
        originalStack.push(5);

        cout << "Original stack: ";
        stack<int> tempStack = originalStack; // Membuat salinan
    stack untuk ditampilkan
        while (!tempStack.empty()) {
            cout << tempStack.top() << " "; // Menampilkan elemen-
    elemen stack
            tempStack.pop(); // Menghapus elemen-elemen yang sudah
    ditampilkan
        }
        cout << endl;

        // Memanggil fungsi untuk membalikkan stack
        reverseStack(originalStack);

        cout << "Reversed stack: ";
        while (!originalStack.empty()) {
            cout << originalStack.top() << " "; // Menampilkan
    elemen-elemen stack yang sudah dibalikkan
            originalStack.pop(); // Menghapus elemen-elemen yang
    sudah ditampilkan
        }
        cout << endl;

        return 0;
}
```

2. Stack using a queue

   Digunakan untuk menyimpan elemen – elemen stack. Fungsi push menambahkan elemen ke dalam stack dengan memindahkan seluruh elemen yang ada kebelakang agar elemen baru dapat di letakan di depan. Fungsi pop menghapus elemen teratas dari stack, sedangkan top

mengembalikan nilai dari elemen teratas stack. Fungsi empty memeriksa apakah stack kosong atau tidak.

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

class StackUsingQueue {
private:
    queue<int> dataQueue; // Queue untuk menyimpan elemen
stack

public:
    // Fungsi untuk menambahkan elemen ke dalam stack
    void push(int value) {
        int size = dataQueue.size(); // Mengambil jumlah
elemen saat ini dalam queue
        dataQueue.push(value); // Menambahkan elemen baru ke
dalam queue

        // Memindahkan elemen-elemen lain ke belakang
        for (int i = 0; i < size; i++) {
            dataQueue.push(dataQueue.front()); // Memasukkan
elemen pertama ke belakang
            dataQueue.pop(); // Menghapus elemen pertama yang
sudah dipindahkan
        }
    }

    // Fungsi untuk menghapus elemen teratas dari stack
    void pop() {
        if (dataQueue.empty()) {
            throw out_of_range("Stack is empty");
        }
        dataQueue.pop(); // Menghapus elemen teratas dari
queue
    }

    // Fungsi untuk mendapatkan nilai dari elemen teratas
dalam stack
    int top() {
        if (dataQueue.empty()) {
            throw out_of_range("Stack is empty");
        }
        return dataQueue.front(); // Mengembalikan nilai dari
elemen teratas dalam queue
    }

    // Fungsi untuk memeriksa apakah stack kosong atau tidak
    bool empty() {
        return dataQueue.empty();
    }
```

```
};

int main() {
    StackUsingQueue stack;
    stack.push(1);
    stack.push(2);
    stack.push(3);

    cout << "Top: " << stack.top() << endl; // Menampilkan
nilai dari elemen teratas dalam stack

    stack.pop(); // Menghapus elemen teratas dari stack

    cout << "Top: " << stack.top() << endl; // Menampilkan
nilai dari elemen teratas dalam stack setelah pop

    return 0;
}
```