

Praktikum Algoritma dan Struktur Data

Queue



Oleh :

Rayhan Elmo Athalah Saputra (5223600027)

Program Studi Sarjana Terapan Teknologi Game

Departemen Teknologi Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2024

1. Queue Menggunakan Linked List

Program ini mengimplementasikan struktur data Queue menggunakan Linked List.

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

// Definisi kelas Queue
class Queue {
private:
    // Struktur Node untuk menyimpan nilai dan pointer ke node
    berikutnya
    struct Node {
        int value; // Nilai dari node
        Node* next; // Pointer ke node berikutnya
        Node(int v, Node* n) : value(v), next(n) {} // Konstruktor
        untuk inisialisasi nilai dan pointer
    };

    Node* head = nullptr; // Pointer ke node pertama (head) dalam
    antrian
    Node* tail = nullptr; // Pointer ke node terakhir (tail) dalam
    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 semua elemen dalam antrian
    void print() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->value << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // Fungsi untuk melihat nilai dari elemen pertama dalam antrian
    tanpa menghapusnya
    int peek() {
        if (empty()) {
```

```

        throw invalid_argument("QueueEmptyException");
    }
    return head->value;
}

// Fungsi untuk menambahkan nilai ke dalam antrian
void enqueue(int value) {
    Node* temp = new Node(value, nullptr); // Membuat node baru
dengan nilai yang diberikan
    if (head == nullptr) { // Jika antrian kosong, node baru
menjadi node pertama dan terakhir
        head = tail = temp;
    }
    else {
        tail->next = temp; // Menambahkan node baru setelah node
terakhir
        tail = temp; // Node baru menjadi node terakhir
    }
    count++; // Menambah jumlah elemen dalam antrian
}

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

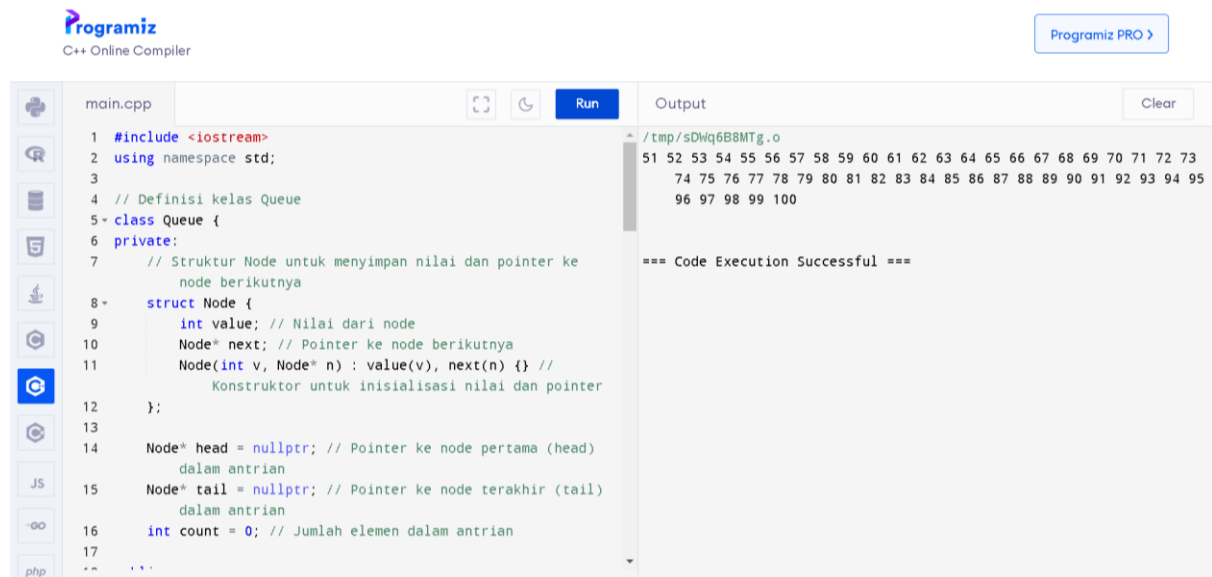
// Fungsi utama program
int main() {
    Queue q; // Membuat objek antrian dari kelas Queue
    for (int i = 1; i <= 100; i++) { // Menambahkan nilai dari 1
hingga 100 ke dalam antrian
        q.enqueue(i);
    }
}

```

```

        for (int i = 1; i <= 50; i++) { // Menghapus 50 elemen pertama
            dari antrian
                q.dequeue();
        }
        q.print(); // Mencetak nilai dari elemen-elemen yang tersisa dalam
        antrian
        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 is divided into two panels: 'main.cpp' on the left and 'Output' on the right. The 'main.cpp' panel contains the following code:

```

1 #include <iostream>
2 using namespace std;
3
4 // Definisi kelas Queue
5 class Queue {
6 private:
7     // Struktur Node untuk menyimpan nilai dan pointer ke
       node berikutnya
8     struct Node {
9         int value; // Nilai dari node
10        Node* next; // Pointer ke node berikutnya
11        Node(int v, Node* n) : value(v), next(n) {} //
        Konstruktor untuk inisialisasi nilai dan pointer
12    };
13
14    Node* head = nullptr; // Pointer ke node pertama (head)
        dalam antrian
15    Node* tail = nullptr; // Pointer ke node terakhir (tail)
        dalam antrian
16    int count = 0; // Jumlah elemen dalam antrian
17
18    ...

```

The 'Output' panel shows the result of the program execution:

```

/tmp/sDwq6B8MTg.o
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100

=== Code Execution Successful ===

```

2. Enqueue pada Queue Linked List

Program ini hanya memuat implementasi metode `enqueue` untuk menambahkan elemen baru ke akhir Queue Linked List.

```

// Fungsi untuk menambahkan nilai ke dalam antrian
void Queue::enqueue(int value) {
    Node* temp = new Node(value, nullptr); // Membuat node baru dengan
    nilai yang diberikan
    if (head == nullptr) { // Jika antrian masih kosong
        head = tail = temp; // Node baru menjadi node pertama dan
        terakhir
    } else { // Jika antrian tidak kosong
        tail->next = temp; // Menambahkan node baru setelah node
        terakhir
        tail = temp; // Node baru menjadi node terakhir
    }
    count++; // Menambah jumlah elemen dalam antrian
}

```

3. Dequeue pada Queue Linked List

Program ini hanya mengandung implementasi metode dequeue yang bertugas menghapus elemen dari awal Queue yang diimplementasikan menggunakan Linked List.

```
// Fungsi untuk menghapus dan mengembalikan nilai pertama dari antrian
int Queue::dequeue() {
    if (empty()) { // Jika antrian kosong
        throw invalid_argument("QueueEmptyException"); // Melemparkan
        pengecualian jika antrian kosong
    }
    int value = head->value; // Mengambil nilai dari elemen pertama
    antrian
    Node* temp = head; // Menyimpan alamat dari elemen pertama antrian
    dalam variabel sementara
    head = head->next; // Menggeser head ke elemen kedua dalam antrian
    delete temp; // Menghapus elemen pertama antrian
    count--; // Mengurangi jumlah elemen dalam antrian
    return value; // Mengembalikan nilai dari elemen pertama yang
    dihapus dari antrian
}
```

4. Implementasi Queue menggunakan 2 Stack

Program ini memanfaatkan dua tumpukan (Stack) untuk mengimplementasikan struktur data Queue.

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

// Deklarasi kelas QueueUsingStack
class QueueUsingStack {
private:
    stack<int> inbox, outbox; // Stack untuk menyimpan elemen dalam
    antrian

public:
    // Fungsi untuk menambahkan elemen ke dalam antrian
    void enqueue(int value) {
        inbox.push(value); // Menambahkan nilai ke dalam stack inbox
    }

    // Fungsi untuk menghapus dan mengembalikan elemen pertama dari
    antrian
```

```

int dequeue() {
    if (outbox.empty()) { // Jika stack outbox kosong
        if (inbox.empty()) { // Jika stack inbox juga kosong
            throw invalid_argument("QueueEmptyException"); //
Melempar pengecualian jika antrian kosong
        }
        // Memindahkan semua elemen dari inbox ke outbox
        while (!inbox.empty()) {
            int value = inbox.top(); // Mengambil nilai dari
elemen teratas inbox
            inbox.pop(); // Menghapus elemen teratas dari inbox
            outbox.push(value); // Menambahkan nilai ke dalam
outbox
        }
    }
    int frontValue = outbox.top(); // Mengambil nilai dari elemen
pertama antrian (elemen teratas outbox)
    outbox.pop(); // Menghapus elemen pertama dari outbox
    return frontValue; // Mengembalikan nilai dari elemen pertama
antrian
}
};

// Fungsi utama program
int main() {
    QueueUsingStack queue; // Membuat objek antrian menggunakan stack
    queue.enqueue(1); // Menambahkan nilai 1 ke dalam antrian
    queue.enqueue(11); // Menambahkan nilai 11 ke dalam antrian
    queue.enqueue(111); // Menambahkan nilai 111 ke dalam antrian
    cout << queue.dequeue() << endl; // Menghapus dan mencetak nilai
pertama dari antrian (Output: 1)
    queue.enqueue(2); // Menambahkan nilai 2 ke dalam antrian
    queue.enqueue(21); // Menambahkan nilai 21 ke dalam antrian
    queue.enqueue(211); // Menambahkan nilai 211 ke dalam antrian
    cout << queue.dequeue() << endl; // Menghapus dan mencetak nilai
pertama dari antrian (Output: 11)
    cout << queue.dequeue() << endl; // Menghapus dan mencetak nilai
pertama dari antrian (Output: 111)
    return 0; // Mengakhiri program
}

```

Programiz
C++ Online Compiler

Programiz PRO >

main.cpp

1 #include <iostream>

2 #include <stack>

3 #include <stdexcept>

4 using namespace std;

5

6 // Deklarasi kelas QueueUsingStack

7 class QueueUsingStack {

8 private:

9 stack<int> inbox, outbox; // Stack untuk menyimpan elemen dalam antrian

10

11 public:

12 // Fungsi untuk menambahkan elemen ke dalam antrian

13 void enqueue(int value) {

14 inbox.push(value); // Menambahkan nilai ke dalam stack inbox

15 }

16

17 // Fungsi untuk menghapus dan mengembalikan elemen pertama dari antrian

18 int dequeue() {

19

20 }

21 }

Output

Clear

/tmp/jIogguv0H7.o

1

11

111

=== Code Execution Successful ===

Tugas

1. Reverse Stack

Program Reverse Stack adalah program yang mengimplementasikan algoritma untuk membalikkan urutan elemen dalam sebuah stack. Langkah-langkah singkat dalam program ini adalah sebagai berikut:

- Program dimulai dengan mendefinisikan sebuah fungsi bernama reverseStack.
- Dalam fungsi reverseStack, terdapat dua langkah utama untuk melakukan reverse:
 - Pertama, elemen-elemen dari stack awalnya disalin ke dalam sebuah queue.
 - Kedua, elemen-elemen tersebut dipindahkan kembali ke dalam stack, sehingga urutannya terbalik.
- Di dalam fungsi main, sebuah stack awalnya diisi dengan beberapa nilai.
- Kemudian, nilai-nilai dari stack tersebut ditampilkan sebagai "Original stack".
- Fungsi reverseStack dipanggil untuk mereverse urutan elemen dalam stack.
- Setelah proses reverse selesai, nilai-nilai stack yang telah direverse ditampilkan sebagai "Reversed stack".

```

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

// Fungsi untuk mereverse stack
void reverseStack(stack<int>& stk) {
    queue<int> q; // Membuat sebuah queue

    // Step 1: Menyalin elemen dari stack ke queue
    while (!stk.empty()) { // Selama stack tidak kosong
        q.push(stk.top()); // Menambahkan elemen teratas stack ke
        dalam queue
        stk.pop(); // Menghapus elemen teratas stack
    }

    // Step 2: Menyalin elemen dari queue kembali ke stack
    while (!q.empty()) { // Selama queue tidak kosong
        stk.push(q.front()); // Menambahkan elemen pertama queue ke
        dalam stack
        q.pop(); // Menghapus elemen pertama queue
    }
}

// Fungsi utama program
int main() {
    stack<int> s; // Membuat sebuah stack

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

    cout << "Original stack: "; // Mencetak pesan untuk stack asli
    stack<int> temp = s; // Membuat salinan stack untuk ditampilkan
    while (!temp.empty()) { // Selama stack salinan tidak kosong
        cout << temp.top() << " "; // Mencetak elemen teratas stack
        salinan
        temp.pop(); // Menghapus elemen teratas stack salinan
    }
    cout << endl;

    // Mereverse stack
    reverseStack(s); // Memanggil fungsi untuk mereverse stack
}

```



```

        cout << "Reversed stack: "; // Mencetak pesan untuk stack yang
telah direverse
        while (!s.empty()) { // Selama stack tidak kosong
            cout << s.top() << " "; // Mencetak elemen teratas stack yang
telah direverse
            s.pop(); // Menghapus elemen teratas stack yang telah
direverse
        }
        cout << endl;

        return 0; // Mengakhiri program
    }
}

```

Programiz
C++ Online Compiler

Programiz PRO >

```

main.cpp
1 #include <iostream>
2 #include <stack>
3 #include <queue>
4 using namespace std;
5
6 // Fungsi untuk mereverse stack
7 void reverseStack(stack<int>& stk) {
8     queue<int> q; // Membuat sebuah queue
9
10    // Step 1: Menyalin elemen dari stack ke queue
11    while (!stk.empty()) { // Selama stack tidak kosong
12        q.push(stk.top()); // Menambahkan elemen teratas
13        stk.pop(); // Menghapus elemen teratas stack
14    }
15
16    // Step 2: Menyalin elemen dari queue kembali ke stack
17    while (!q.empty()) { // Selama queue tidak kosong
18        stk.push(q.front()); // Menambahkan elemen pertama
19        q.pop(); // Menghapus elemen pertama queue
20    }
21 }

```

Output

```

/tmp/6JqVe6EOUk.o
Original stack: 5 4 3 2 1
Reversed stack: 1 2 3 4 5

=== Code Execution Successful ===

```

2. Josephus problem:

Masalah Josephus adalah teka-teki matematis yang melibatkan penyisihan orang-orang dalam sebuah lingkaran dan proses penghapusan mereka secara berurutan sampai hanya tersisa satu orang. Ini dinamai dari Flavius Josephus, seorang sejarawan Yahudi-Romawi kuno yang dikatakan telah menggunakan metode ini untuk menghindari eksekusi oleh tentara Romawi pada masa pemerintahan Kaisar Vespasianus.

Berikut adalah langkah-langkah umum untuk menyelesaikan masalah Josephus:

- Menentukan jumlah orang (n) dan langkah (k): Pertama, Anda harus menentukan berapa banyak orang yang ada dalam lingkaran (n) dan

berapa banyak langkah yang harus diambil sebelum setiap orang dihapus (k).

- Membuat lingkaran: Orang-orang tersebut diatur dalam sebuah lingkaran.
- Memilih langkah awal: Biasanya, Anda mulai dari orang pertama dalam lingkaran.
- Menghapus orang: Mulai dari orang yang dipilih pada langkah sebelumnya, hitung ke langkah ke-k dan hapus orang yang ditemukan pada langkah tersebut.
- Mengulangi proses: Terus lakukan langkah-langkah 4 sampai hanya tersisa satu orang dalam lingkaran.
- Menentukan pemenang: Orang terakhir yang tersisa adalah pemenangnya, yang selamat dari penghapusan.

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

int josephus(int n, int k) {
    list<int> people;

    // Membuat daftar orang dari 1 hingga n
    for (int i = 1; i <= n; i++) {
        people.push_back(i);
    }

    auto it = people.begin();
    while (people.size() > 1) {
        // Memindahkan iterator k-1 langkah
        for (int i = 0; i < k - 1; i++) {
            it++;
            if (it == people.end()) {
                it = people.begin();
            }
        }

        // Menghapus orang saat ini
        auto next = it;
        next++;
        if (next == people.end()) {
            next = people.begin();
        }
        it = people.erase(it);
        it = next;
    }
}
```

```

    }

    return people.front();
}

int main() {
    int n, k;
    cout << "Masukkan jumlah orang: ";
    cin >> n;
    cout << "Masukkan nilai k: ";
    cin >> k;

    int survivor = josephus(n, k);
    cout << "Orang yang selamat adalah: " << survivor << endl;

    return 0;
}

```

Programiz
C++ Online Compiler

Programiz PRO >

main.cpp
Run

```

1 #include <iostream>
2 #include <list>
3 using namespace std;
4
5 int josephus(int n, int k) {
6     list<int> people;
7
8     // Membuat daftar orang dari 1 hingga n
9     for (int i = 1; i <= n; i++) {
10         people.push_back(i);
11     }
12
13     auto it = people.begin();
14     while (people.size() > 1) {
15         // Memindahkan iterator k-1 langkah
16         for (int i = 0; i < k - 1; i++) {
17             it++;
18             if (it == people.end()) {
19                 it = people.begin();
20             }
21         }
22     }

```

Output
Clear

```

/tmp/xC7jK2tY2.o
Masukkan jumlah orang:

```

main.cpp

Run

Clear

```
1 #include <iostream>
2 #include <list>
3 using namespace std;
4
5 int josephus(int n, int k) {
6     list<int> people;
7
8     // Membuat daftar orang dari 1 hingga n
9     for (int i = 1; i <= n; i++) {
10         people.push_back(i);
11     }
12
13     auto it = people.begin();
14     while (people.size() > 1) {
15         // Memindahkan iterator k-1 langkah
16         for (int i = 0; i < k - 1; i++) {
17             it++;
18             if (it == people.end()) {
19                 it = people.begin();
20             }
21         }
22     }
23 }
```

Output

```
/tmp/xC7jK2tY2.o
Masukkan jumlah orang: 3
Masukkan nilai k: 12
Orang yang selamat adalah: 1

=== Code Execution Successful ===
```