**LAPORAN PRAKTIKUM**
**Modul 08**
**“Queue”**

**Disusun Oleh:**
**Ganesha Rahman Gibran -2211104058**
**Kelas S1SE-07-02**

**Dosen :**
**Wahyu Andi Saputra, S.Pd., M.Eng.**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**
**2024**

## A. TUJUAN PRAKTIKUM

a. Mahasiswa mampu menjelaskan definisi dan konsep dari queue
b. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
c. Mahasiswa mampu menerapkan operasi tampil data pada queue

## B. DASAR TEORI

### Queue: Struktur Data FIFO (First-In First-Out)

Queue adalah salah satu struktur data yang menerapkan prinsip FIFO (First-In First-Out). Artinya, elemen yang pertama kali masuk ke dalam queue akan menjadi elemen pertama yang dikeluarkan. Konsep ini serupa dengan antrian dalam kehidupan sehari-hari, di mana orang yang pertama kali tiba akan dilayani lebih dahulu.

Implementasi queue dapat dilakukan menggunakan array atau linked list. Struktur data ini memiliki dua pointer utama, yaitu *front* dan *rear*. Pointer *front* menunjuk ke elemen pertama dalam queue, sementara pointer *rear* menunjuk ke elemen terakhir.

### Perbandingan Queue dengan Stack

Perbedaan mendasar antara queue dan stack terletak pada aturan pengelolaan elemen. Pada stack, elemen ditambahkan dan dihapus dari ujung yang sama, yaitu *top*. Hal ini mengikuti prinsip LIFO (Last-In First-Out), di mana elemen terakhir yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Contoh analogi stack adalah tumpukan piring, di mana piring yang terakhir ditambahkan akan diambil terlebih dahulu.

Sebaliknya, queue memungkinkan elemen ditambahkan di satu ujung (belakang, atau *rear*) dan dihapus dari ujung lainnya (depan, atau *front*). Proses ini sesuai dengan prinsip FIFO, di mana elemen pertama yang masuk akan keluar lebih dulu. Pada queue, operasi penambahan elemen disebut **enqueue**, sedangkan penghapusan elemen disebut **dequeue**. Saat melakukan **enqueue**, elemen baru ditempatkan di belakang queue, dan saat **dequeue**, elemen di depan dihapus sehingga pointer *front* bergeser ke elemen berikutnya. Contoh penerapan queue dalam kehidupan nyata adalah antrian kasir, di mana pelanggan yang pertama tiba akan dilayani lebih dulu.

### Operasi pada Queue

1. **enqueue()**: Menambahkan elemen baru ke belakang queue.
2. **dequeue()**: Menghapus elemen di bagian depan queue.
3. **peek()**: Mengambil elemen di depan queue tanpa menghapusnya.
4. **isEmpty()**: Memeriksa apakah queue kosong.
5. **isFull()**: Memeriksa apakah queue telah penuh.

6. **size()**: Menghitung jumlah elemen dalam queue.

Queue merupakan struktur data yang sangat berguna untuk situasi yang membutuhkan pengolahan data secara berurutan sesuai urutan masuk

## C. GUIDED 1

Sourcecode

```
#include <iostream>
#define MAX 100

using namespace std;

class Queue {
private:
    int front, rear;
    int arr[MAX];
public:

    Queue() {
        front = -1;
        rear = -1;
    }


    bool isFull() {
        return rear == MAX - 1;
    }


    bool isEmpty() {
        return front == -1 || front > rear;
    }


    void enqueue(int x) {
        if (isFull()) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        arr[++rear] = x;
    }


    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }

    int peek() {
        if (!isEmpty()) {
            return arr[front];
        }
        cout << "Queue is empty\n";
        return -1;
    }
```

```
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << "\n";
    }
};


int main() {
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << "Queue elements: ";
    q.display();

    cout << "Front element: " << q.peek() << "\n";

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

Output

```
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output> & .\'guided
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output>
```

## GUIDED 2

Sourcecode

```
#include <iostream>

using namespace std;

// Node untuk setiap elemen Queue
class Node {
public:
    int data;      // Data elemen
    Node* next;    // Pointer ke node berikutnya

    // Konstruktor untuk Node
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
```

```
// Kelas Queue menggunakan linked list
class Queue {
private:
    Node* front; // Pointer ke elemen depan Queue
    Node* rear;  // Pointer ke elemen belakang Queue

public:
    // Konstruktor Queue
    Queue() {
        front = rear = nullptr;
    }

    // Mengecek apakah Queue kosong
    bool isEmpty() {
        return front == nullptr;
    }

    // Menambahkan elemen ke Queue
    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {
            front = rear = newNode; // Jika Queue kosong
            return;
        }
        rear->next = newNode; // Tambahkan node baru ke belakang
        rear = newNode;       // Perbarui rear
    }

    // Menghapus elemen dari depan Queue
    void dequeue() {
        if (isEmpty()) {
           cout << "Queue Underflow\n";
            return;
        }
        Node* temp = front;      // Simpan node depan untuk dihapus
        front = front->next;     // Pindahkan front ke node berikutnya
        delete temp;             // Hapus node lama
        if (front == nullptr)    // Jika Queue kosong, rear juga harus null
            rear = nullptr;
    }

    // Mengembalikan elemen depan Queue tanpa menghapusnya
    int peek() {
        if (!isEmpty()) {
            return front->data;
        }
       cout << "Queue is empty\n";
        return -1; // Nilai sentinel
    }

    // Menampilkan semua elemen di Queue
    void display() {
        if (isEmpty()) {
           cout << "Queue is empty\n";
            return;
        }
        Node* current = front; // Mulai dari depan
        while (current) {      // Iterasi sampai akhir
           cout << current->data << " ";
            current = current->next;
        }
       cout << "\n";
```

```
    }
};

// Fungsi utama untuk menguji Queue
int main() {
    Queue q;

    // Menambahkan elemen ke Queue
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Menampilkan elemen di Queue
   cout << "Queue elements: ";
    q.display();

    // Menampilkan elemen depan
   cout << "Front element: " << q.peek() << "\n";

    // Menghapus elemen dari depan Queue
    q.dequeue();
   cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

Output

```
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output> & .\
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output>
```

**GUIDED 3**

Sourcecode

```
#include<iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian
int front = 0; // Penanda antrian
int back = 0; // Penanda
string queueTeller[5]; // Fungsi pengecekan

bool isFull() { // Pengecekan antrian penuh atau tidak
if (back == maksimalQueue) { return true; // =1
} else {
return false;
}
}

bool isEmpty() { // Antriannya kosong atau tidak
if (back == 0) { return true;
} else {
return false;
}
}
```

```
void enqueueAntrian(string data) { // Fungsi menambahkan antrian
if (isFull()) {
cout << "Antrian penuh" << endl;
} else {
if (isEmpty()) { // Kondisi ketika queue kosong
queueTeller[0] = data; front++;
back++;
} else { // Antrianya ada isi queueTeller[back] = data; back++;
}
}
}

void dequeueAntrian() { // Fungsi mengurangi antrian
if (isEmpty()) {
cout << "Antrian kosong" << endl;
} else {
for (int i = 0; i < back; i++) { queueTeller[i] = queueTeller[i + 1];
}
back--;
}
}

int countQueue() { // Fungsi menghitung banyak antrian
return back;
}

void clearQueue() { // Fungsi menghapus semua antrian
if (isEmpty()) {
cout << "Antrian kosong" << endl;
} else {
for (int i = 0; i < back; i++) { queueTeller[i] = "";
}
back = 0;
front = 0;
}
}

void viewQueue() { // Fungsi melihat antrian
cout << "Data antrian teller:" << endl; for (int i = 0; i < maksimalQueue; i++)
{
if (queueTeller[i] != "") {
cout << i + 1 << ". " << queueTeller[i] <<

endl;


} else {
cout << i + 1 << ". (kosong)" << endl;

}
}
}

int main() {
enqueueAntrian("Andi");

enqueueAntrian("Maya");

viewQueue();
cout << "Jumlah antrian = " << countQueue() << endl;
```

```
dequeueAntrian();
viewQueue();
cout << "Jumlah antrian = " << countQueue() << endl;

clearQueue();
viewQueue();
cout << "Jumlah antrian = " << countQueue() << endl;

return 0;
}
```

Output

```
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output> & .\'g
Data antrian teller:
1. Andi
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Guided\output>
```

## D. UNGUIDED

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadi linked list

   Input :

```
nclude <iostream>
ing namespace std;

uct Node {
  string data;
  Node* next;


de* front = nullptr;
de* rear = nullptr;

ol isEmpty() {
  return front == nullptr;


id enqueue(string data) {
  Node* newNode = new Node();
  newNode->data = data;
  newNode->next = nullptr;
```

```
  if (isEmpty()) {
      front = rear = newNode;
  } else {
      rear->next = newNode;
      rear = newNode;
  }


id dequeue() {
  if (isEmpty()) {
      cout << "Antrian kosong!" << endl;
  } else {
      Node* temp = front;
      front = front->next;
      delete temp;

      if (front == nullptr) {
          rear = nullptr;
      }
  }


id viewQueue() {
  if (isEmpty()) {
      cout << "Antrian kosong!" << endl;
  } else {
      Node* temp = front;
      cout << "Data antrian:" << endl;
      while (temp != nullptr) {
          cout << temp->data << endl;
          temp = temp->next;
      }
  }


t countQueue() {
  int count = 0;
  Node* temp = front;
  while (temp != nullptr) {
      count++;
      temp = temp->next;
  }
  return count;


id clearQueue() {
  while (!isEmpty()) {
      dequeue();
  }


t main() {
  enqueue("Andi");
  enqueue("Maya");
  viewQueue();
  cout << "Jumlah antrian = " << countQueue() << endl;

  dequeue();
  viewQueue();
  cout << "Jumlah antrian = " << countQueue() << endl;

  clearQueue();
```

```
 viewQueue();
 cout << "Jumlah antrian = " << countQueue() << endl;

 return 0;
}
```

Output :

```
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Unguided\output> & .\
Data antrian:
Andi
Maya
Jumlah antrian = 2
Data antrian:
Maya
Jumlah antrian = 1
Antrian kosong!
Jumlah antrian = 0
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Unguided\output>
```

2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM Mahasiswa

Input :

```
nclude <iostream>
ing namespace std;

 Struktur Node untuk Linked List
uct Node {
  string nama;
  string nim;
  Node* next;


 Pointer front dan rear
de* front = nullptr;
de* rear = nullptr;

 Fungsi mengecek apakah Queue kosong
ol isEmpty() {
  return front == nullptr;


 Fungsi menambahkan elemen ke Queue
id enqueue(string nama, string nim) {
  Node* newNode = new Node();
  newNode->nama = nama;
  newNode->nim = nim;
  newNode->next = nullptr;

  if (isEmpty()) {
      front = rear = newNode;
  } else {
      rear->next = newNode;
      rear = newNode;
  }


 Fungsi menghapus elemen dari Queue
id dequeue() {
  if (isEmpty()) {
      cout << "Antrian kosong!" << endl;
```

```
 } else {
     Node* temp = front;
     front = front->next;
     delete temp;

     if (front == nullptr) {
         rear = nullptr;
     }
 }


Fungsi menampilkan elemen di Queue
id viewQueue() {
 if (isEmpty()) {
     cout << "Antrian kosong!" << endl;
 } else {
     Node* temp = front;
     cout << "Data antrian:" << endl;
     while (temp != nullptr) {
         cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
         temp = temp->next;
     }
 }


t main() {
 int pilihan;
 do {
     cout << "\nMenu Queue:" << endl;
     cout << "1. Tambah Antrian" << endl;
     cout << "2. Hapus Antrian" << endl;
     cout << "3. Lihat Antrian" << endl;
     cout << "4. Keluar" << endl;
     cout << "Pilih: ";
     cin >> pilihan;

     if (pilihan == 1) {
         string nama, nim;
         cout << "Masukkan Nama: ";
         cin.ignore();
         getline(cin, nama);
         cout << "Masukkan NIM: ";
         getline(cin, nim);
         enqueue(nama, nim);
     } else if (pilihan == 2) {
         dequeue();
     } else if (pilihan == 3) {
         viewQueue();
     }
 } while (pilihan != 4);

 return 0;
}
```

Output :

```
ahman_Gibran_SE-06-02\08_Queue\Unguided\output'
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\08_Queue\Unguided\output>

Menu Queue:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan Nama: Andi
Masukkan NIM: 1111

Menu Queue:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan Nama: Maya
Masukkan NIM: 2222

Menu Queue:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 3
Data antrian:
Nama: Andi, NIM: 1111
Nama: Maya, NIM: 2222
```

3. Modifikasi program pada soal 1 sehingga mahasiswa dapat diprioritaskan berdasarkan NIM (NIM yang lebih kecil didahulukan pada saat output).

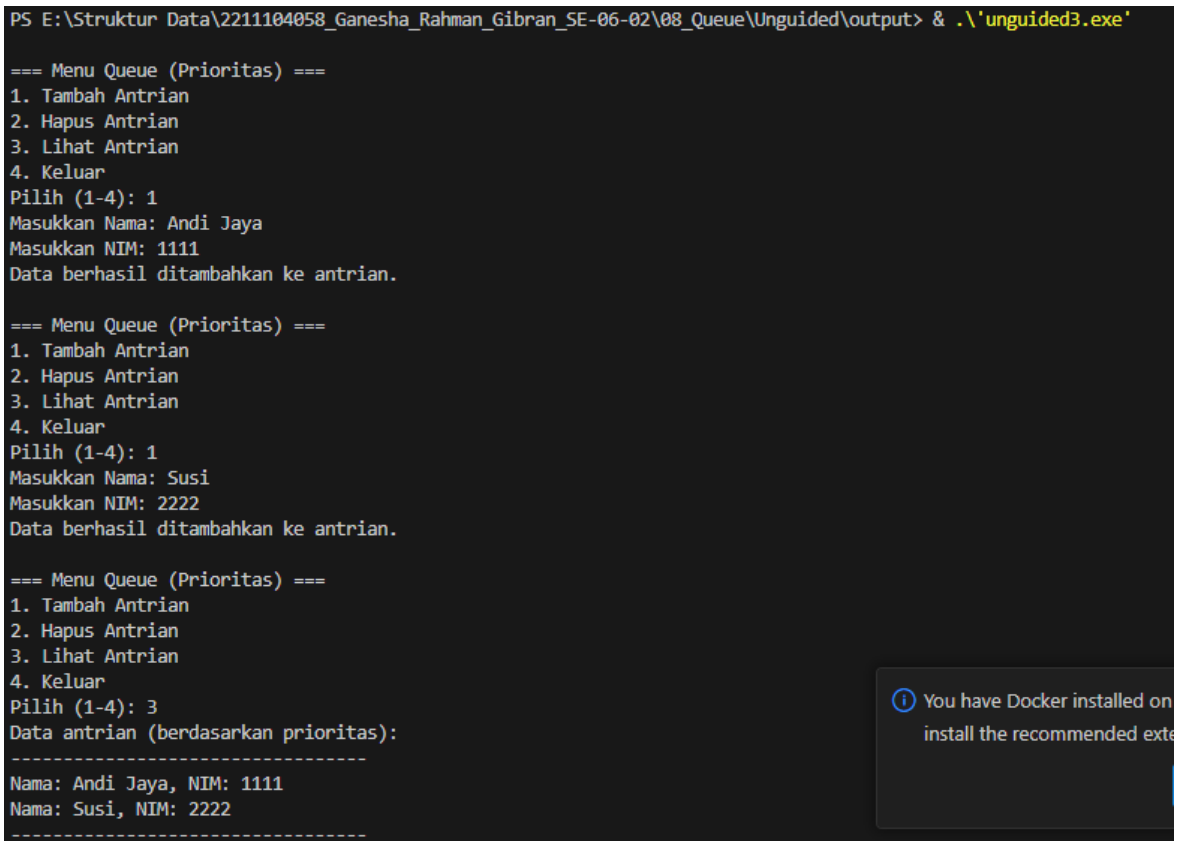
Input :

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

// Struktur Node untuk Linked List
struct Node {
    string nama;
    string nim;
    Node* next;
};

// Pointer front dan rear
Node* front = nullptr;
Node* rear = nullptr;

// Fungsi mengecek apakah Queue kosong
bool isEmpty() {
    return front == nullptr;
}

// Fungsi menambahkan elemen dengan prioritas berdasarkan NIM
void enqueueWithPriority(const string& nama, const string& nim) {
    Node* newNode = new Node();
```

```
 newNode->nama = nama;
 newNode->nim = nim;
 newNode->next = nullptr;

 if (isEmpty() || nim < front->nim) {
     newNode->next = front;
     front = newNode;
     if (rear == nullptr) {
         rear = newNode;
     }
 } else {
     Node* temp = front;
     while (temp->next != nullptr && temp->next->nim < nim) {
         temp = temp->next;
     }
     newNode->next = temp->next;
     temp->next = newNode;
     if (newNode->next == nullptr) {
         rear = newNode;
     }
 }
 cout << "Data berhasil ditambahkan ke antrian.\n";


Fungsi menghapus elemen dari Queue
id dequeue() {
 if (isEmpty()) {
     cout << "Antrian kosong! Tidak ada yang dihapus.\n";
 } else {
     Node* temp = front;
     cout << "Menghapus: Nama: " << temp->nama << ", NIM: " << temp->nim <<
 endl;
     front = front->next;
     delete temp;

     if (front == nullptr) {
         rear = nullptr;
     }
 }


Fungsi menampilkan elemen di Queue
id viewQueue() {
 if (isEmpty()) {
     cout << "Antrian kosong!\n";
 } else {
     Node* temp = front;
     cout << "Data antrian (berdasarkan prioritas):\n";
     cout << "---------------------------------\n";
     while (temp != nullptr) {
         cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
         temp = temp->next;
     }
     cout << "---------------------------------\n";
 }


Fungsi untuk membersihkan seluruh queue
id clearQueue() {
 while (!isEmpty()) {
     dequeue();
 }
```

```
 cout << "Semua data dalam antrian telah dihapus.\n";


t main() {
 int pilihan;
 do {
     cout << "\n=== Menu Queue (Prioritas) ===\n";
     cout << "1. Tambah Antrian\n";
     cout << "2. Hapus Antrian\n";
     cout << "3. Lihat Antrian\n";
     cout << "4. Keluar\n";
     cout << "Pilih (1-4): ";
     cin >> pilihan;

     // Validasi input agar tidak terjadi infinite loop
     if (cin.fail()) {
         cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         cout << "Input tidak valid! Masukkan angka antara 1 hingga 4.\n";
         continue;
     }

     switch (pilihan) {
         case 1: {
             string nama, nim;
             cout << "Masukkan Nama: ";
             cin.ignore(); // Membersihkan buffer sebelum menerima input
string
             getline(cin, nama);
             cout << "Masukkan NIM: ";
             getline(cin, nim);
             enqueueWithPriority(nama, nim);
             break;
         }
         case 2:
             dequeue();
             break;
         case 3:
             viewQueue();
             break;
         case 4:
             cout << "Keluar dari program. Terima kasih!\n";
             clearQueue();
             break;
         default:
             cout << "Pilihan tidak valid! Masukkan angka antara 1 hingga
4.\n";
             break;
     }
 } while (pilihan != 4);

 return 0;
```

Output :




Noted : Untuk data mahasiswa dan nim dimasukan oleh user

## E. KESIMPULAN

Queue adalah struktur data yang bekerja dengan prinsip FIFO (First-In First-Out), di mana elemen yang pertama masuk akan menjadi elemen pertama yang keluar. Dengan implementasi menggunakan array atau linked list, queue memiliki dua pointer utama, yaitu *front* untuk elemen pertama dan *rear* untuk elemen terakhir. Berbeda dengan stack yang menggunakan prinsip LIFO (Last-In First-Out), queue memungkinkan penambahan elemen melalui operasi **enqueue** di belakang dan penghapusan elemen melalui operasi **dequeue** di depan. Operasi tambahan seperti **peek**, **isEmpty**, dan **size** mendukung pengelolaan data secara efisien, membuat queue menjadi solusi ideal untuk antrian atau proses berurutan dalam kehidupan sehari-hari maupun pemrograman.