**MINIMISASI LOGIC**

**LAPORAN TUGAS BESAR**

Sebagai salah satu bagian dari Tugas Besar mata kuliah Pemecahan Masalah dengan C (EL2008) Kelas 1 pada Semester IV Tahun Akademik 2021/2022

oleh

Adro Anra Purnama 13220005

Surya Dharma 13220027

Fariz Iftikhar Falakh 13220029

Senggani Fatah Sedayu 13220035

A picture containing text

Description automatically generated

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

**Daftar Isi**

[**Laporan Inti** 1](#_Toc103881390)

[**A.** **Studi Pustaka** 1](#_Toc103881391)

[**B.** **Spesifikasi Program** 2](#_Toc103881392)

[*1)* *Flowchart* 3](#_Toc103881393)

[*2)* *Data Flow Diagram* 13](#_Toc103881394)

[**D.** **Source Code** 14](#_Toc103881395)

[**Kesimpulan dan Lesson Learned** 25](#_Toc103881396)

[**Pembagian Tugas dalam Kelompok** 26](#_Toc103881397)

[**Link Repository** 26](#_Toc103881398)

[**Referensi** 27](#_Toc103881399)

# **Laporan Inti**

1. **Studi Pustaka**

Minimisasi logika adalah suatu proses untuk mencari persamaan yang lebih sederhana dari suatu rangkaian logika dengan syarat tertentu. Tujuan dari minimisasi logika adalah mengurangi banyaknya gerbang logika atau sirkuit yang digunakan supaya proses lebih sederhana dan penyelesaian persamaan memakan lebih sedikit waktu.

Metode minimisasi logika yang digunakan oleh kelompok ini adalah metode tabular atau metode Quine-McCluskey. Metode ini dipilih karena pembuatan kodenya lebih mudah dibandingkan dengan metode Karnaugh Map yang lebih bergantung pada visual. Metode tabular juga lebih mudah digunakan dalam menyelesaikan masalah dengan variabel yang banyak dibandingkan dengan Karnaugh Map. Dalam minimisasi logika, metode tabular menggunakan sistem eliminasi di mana pengaplikasian algoritma dapat menggunakan rekursi tanpa mengubah fungsi yang telah ada. Dibandingkan dengan Karnaugh Map yang perlu mengubah letak urutan dari bentuk tabel tergantung dengan banyak variabel dan menjadi lebih kompleks setelah ada 4 variabel, metode tabular lebih mudah digunakan.

Langkah dalam menggunakan metode tabular adalah pertama susun minterm yang diberikan dari yang terkecil hingga yang terbesar dan buat grup berdasarkan jumlah bilangan satu yang ada dalam representasi binernya. Sehiangga, akan ada maksimal grup jika ada variabel Boolean dalam fungsi Boolean atau bit dalam ekuivalen biner dari minterm

Kedua, bandingkan minterm yang ada pada suatu grup dengan minterm lain yang ada pada grup selanjutnya. Jika perbedaan antara minterm satu dengan yang lainnya hanya satu bit, maka gabungkan dua minterm tersebut menjadi sebuah term baru. Tempatkan simbol atau di bit yang berbeda dan biarkan bit lainnya. Selanjutnya ulangi langkah kedua dengan term yang baru terbuat hingga kita mendapatkan semua *prime implicants*.

Ketiga, buat *prime implicant* table. Tabel tersebut terdiri dari kumpulan baris dan kolom. *Prime implicant*s disusun sebagai baris dari tabel dan minterm disusun sebagai kolom dari tabel. Tempatkan '1' di sel yang bisa direpresentasikan oleh *Prime implicant*s serta minterm pada sel tersebut.

Keempat, cari *prime implicant* penting dengan mengamati tiap kolom. Apabila minterm hanya tercakup oleh satu *prime implicant*, maka *prime implicant* tersebut penting. *Prime implicant* tersebut akan menjadi bagian dari penyederhanaan fungsi boolean.

Terakhir, Kurangi tabel *prime implicant* dengan menghilangkan baris dari tiap *prime implicant* penting dan kolom yang bersesuaian dengan minterm yang telah tercakup didalam *prime implicant* penting tersebut. Ulangi langkah keempat untuk mengurangi tabel *prime implicant*. Proses dihentikan ketika semua minterm dari fungsi boolean yang diberikan telah tercakup.

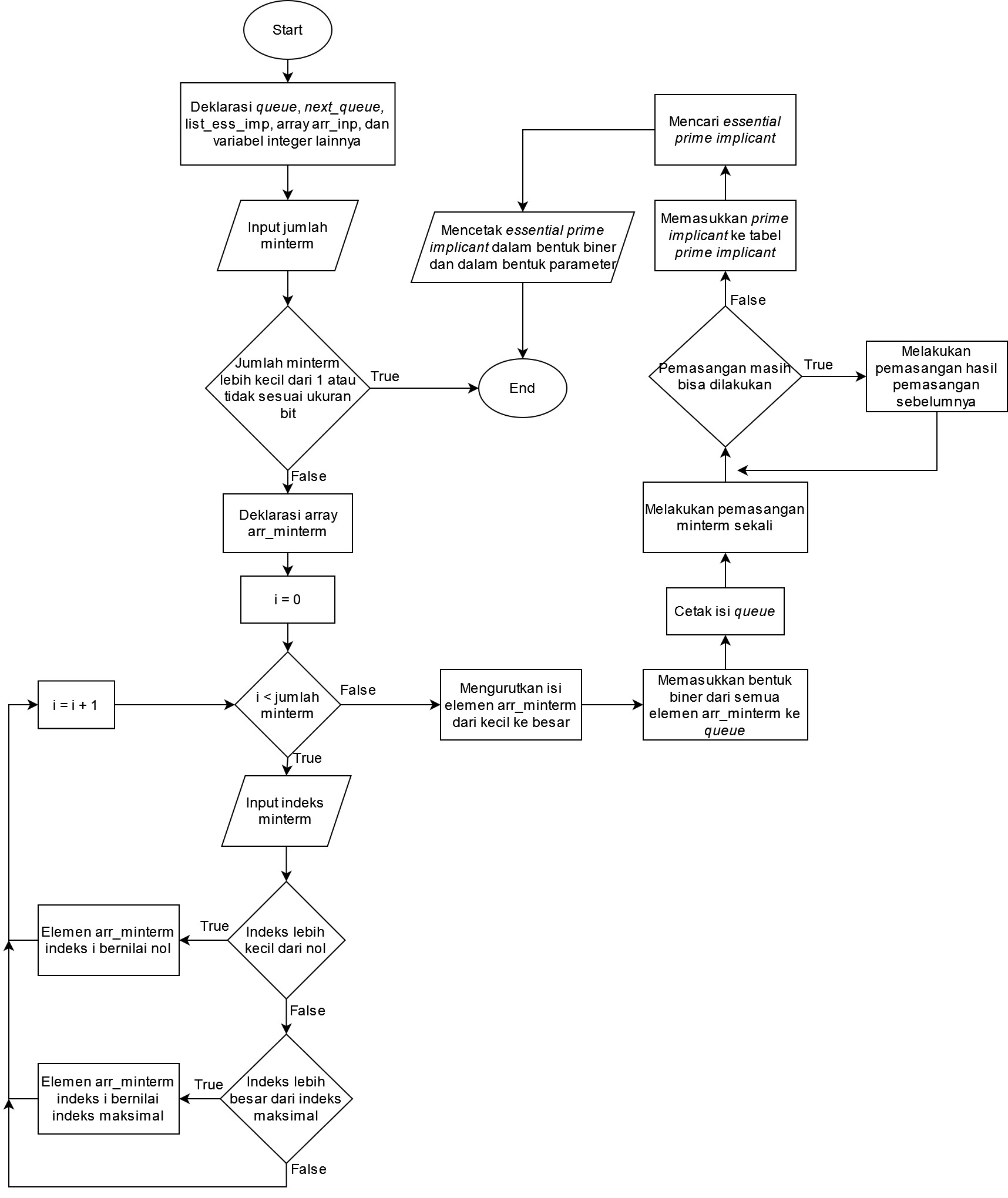
1. **Spesifikasi Program**

Spesifikasi program minimisasi adalah:

1. Program menerima input jumlah minterm dan indeks minterm
2. Program menampilkan hasil minimisasi berdasakran minterm yang diberikan
3. Program hanya bekerja untuk ukuran 2 – 8 bit
4. Ukuran bit hanya bisa diganti dengan mengganti definisi bitsSize pada *header* kode
5. Program tidak menerima minterm *don’t care* dan menganggap tidak ada minterm *don’t care*
6. **Flowchart dan Data Flow Diagram**

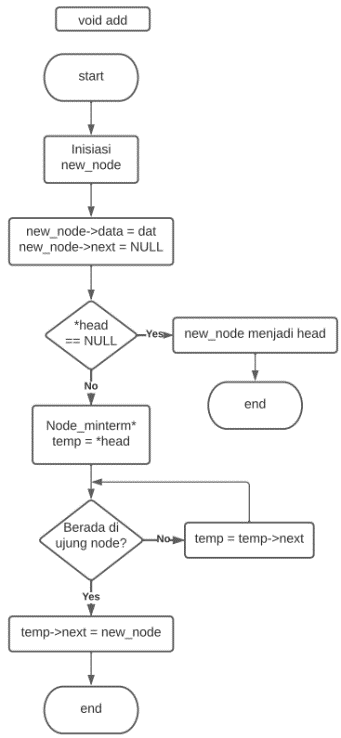
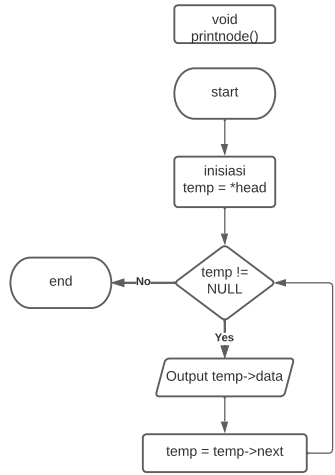
### *Flowchart*

* 1. *Flowchart* algoritma

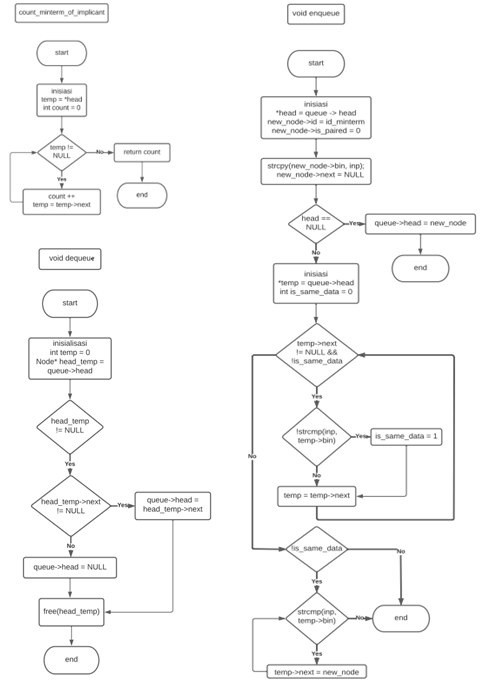


#### Gambar 1. *Flowchart* program secara keseluruhan

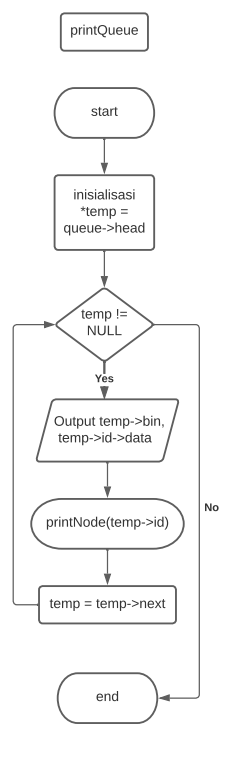
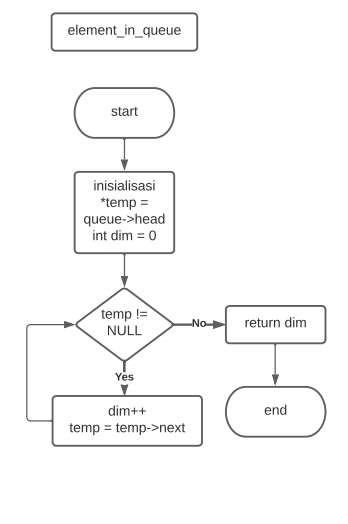
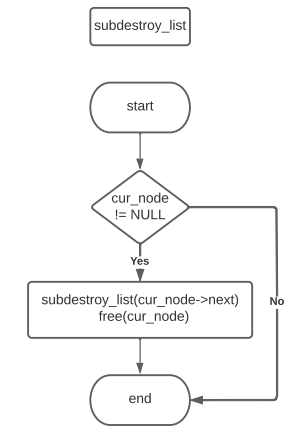
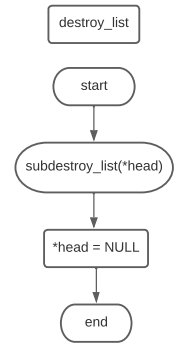
* 1. *Flowchart* fungsi dan prosedur yang digunakan



#### Gambar 2. Flowchart prosedur add dan prosedur printNode



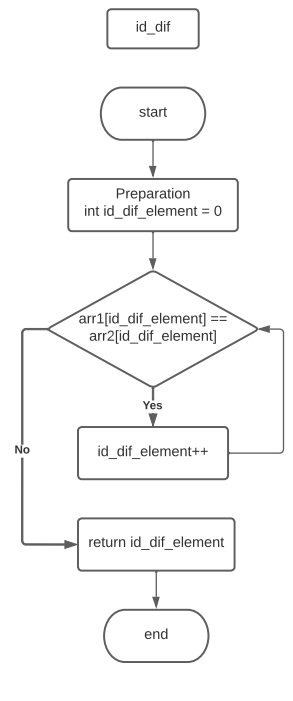
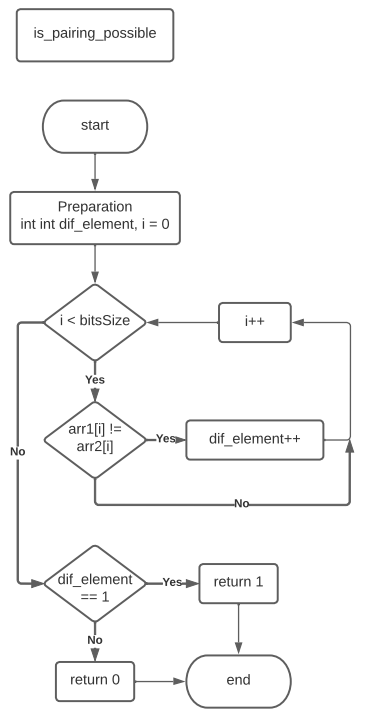
#### Gambar 3. Flowchart fungsi count\_minterm\_of\_implicant, prosedur enqueue, dan prosedur dequeue

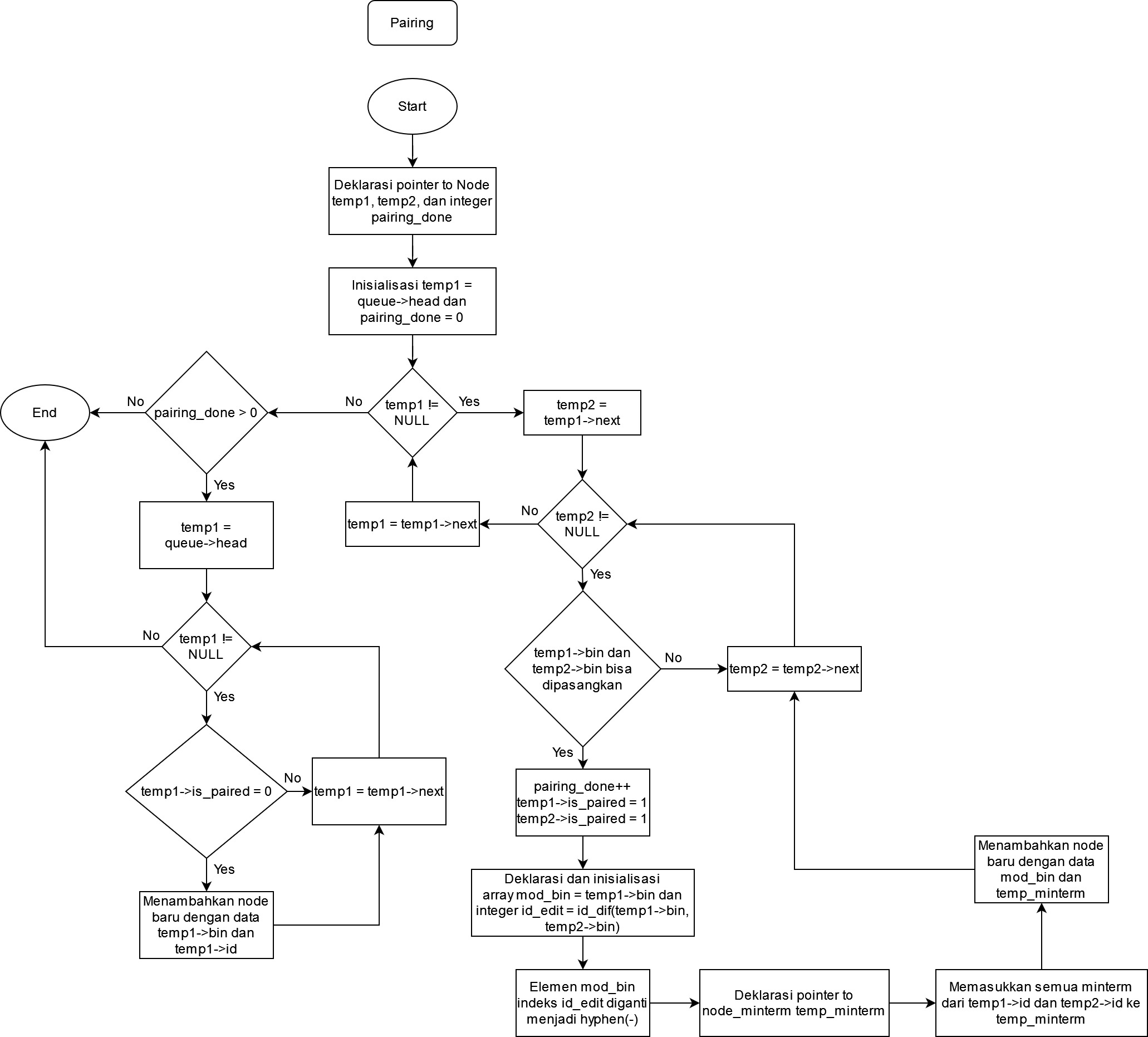


#### Gambar 4. Flowchart prosedur printQueue, fungsi element\_in\_queue, prosedur subdestroy\_list, dan prosedur destroy\_list

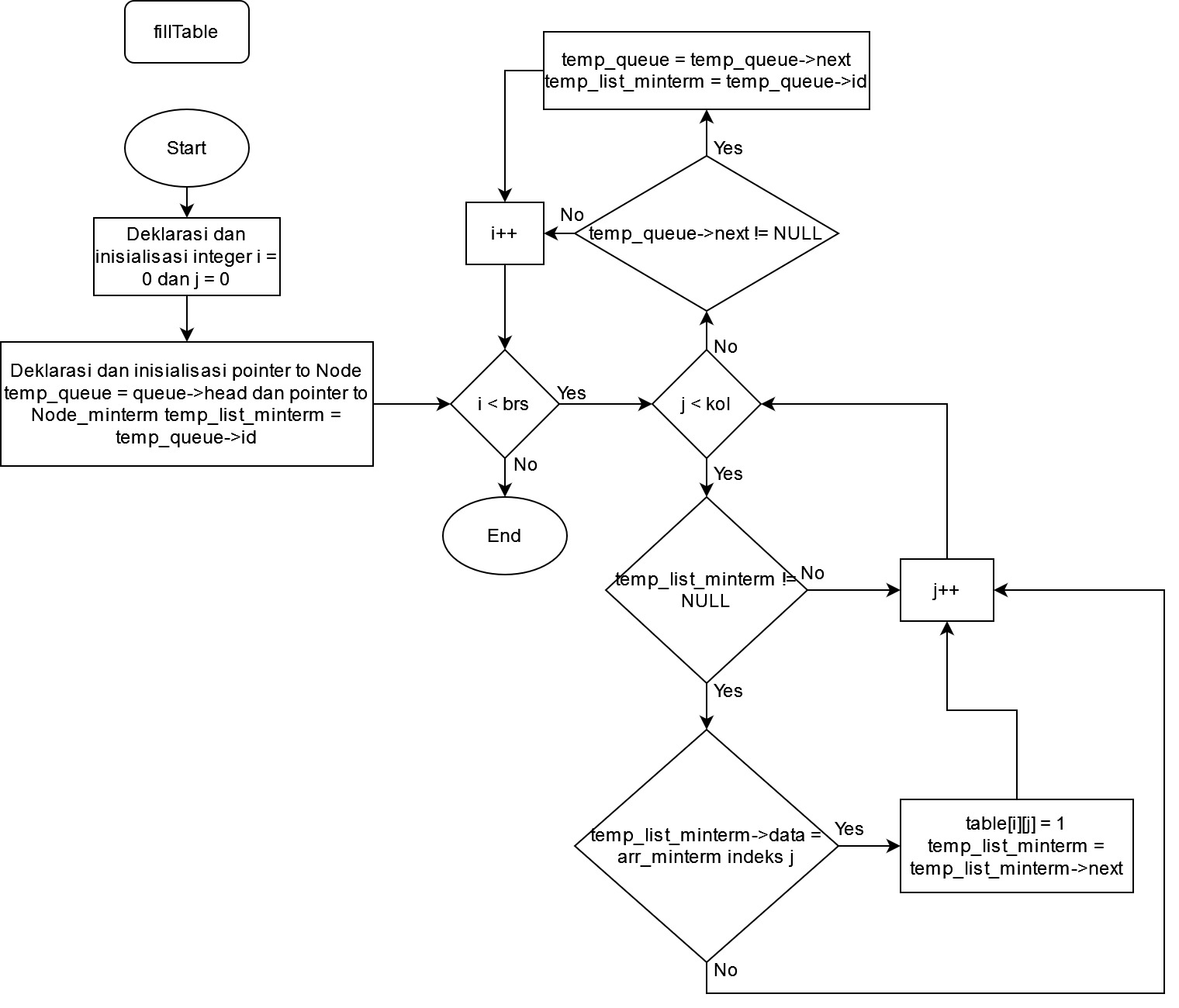
#### Gambar 5. *Flowchart* prosedur destroy\_implicant\_list dan prosedur dec2bin

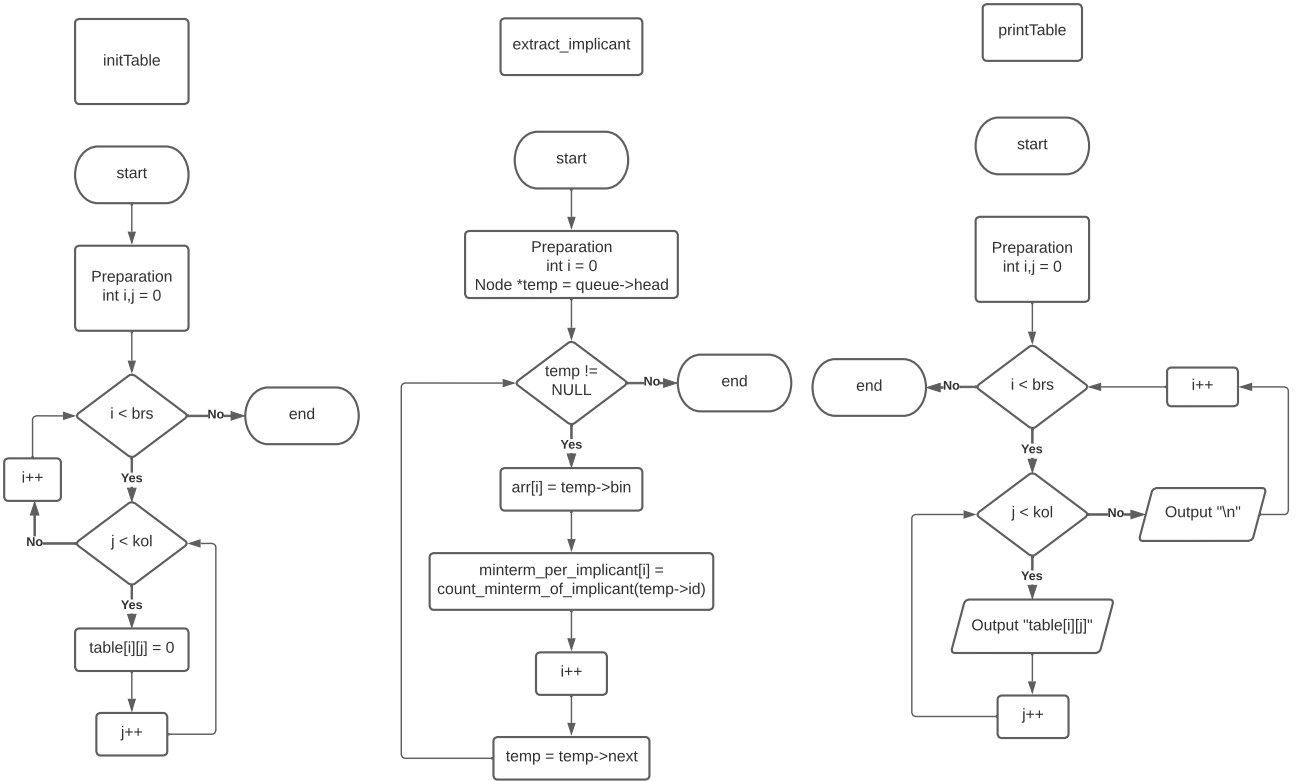
Gambar 6. Flowchart fungsi id\_dif dan fungsi is\_pairing\_possible

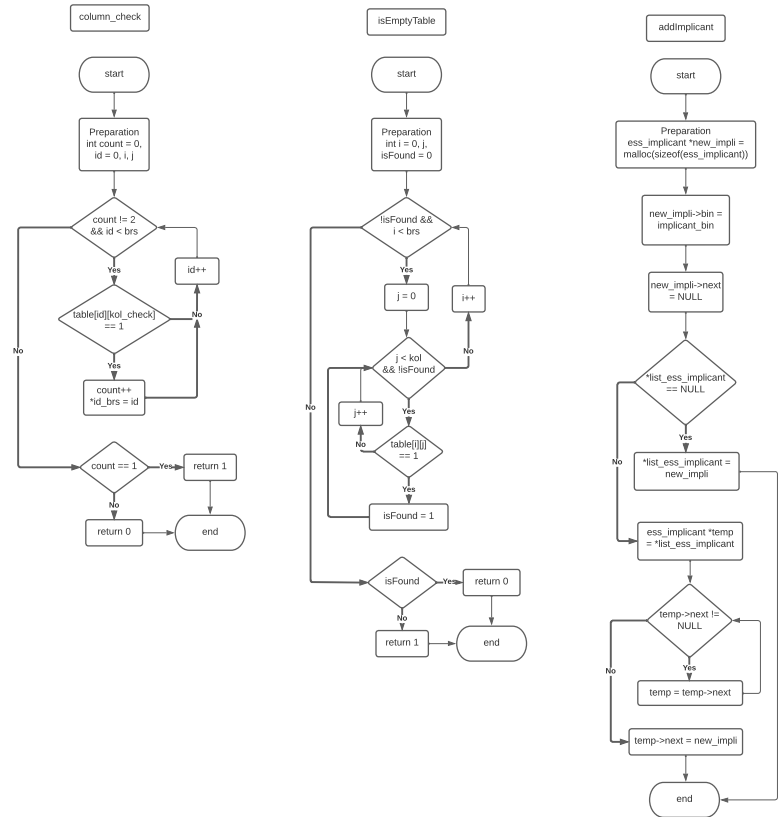


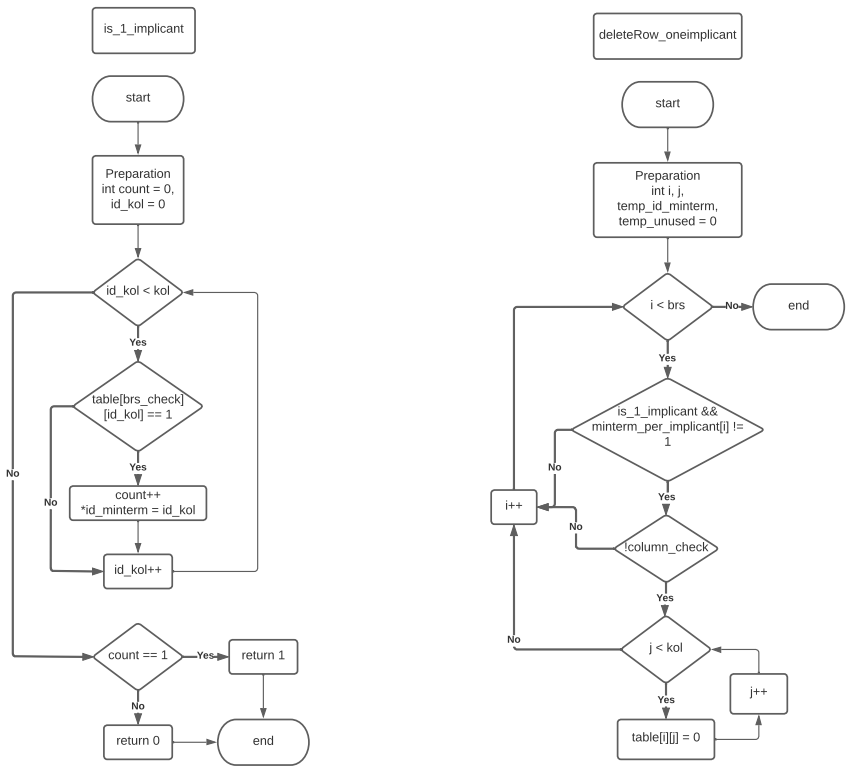


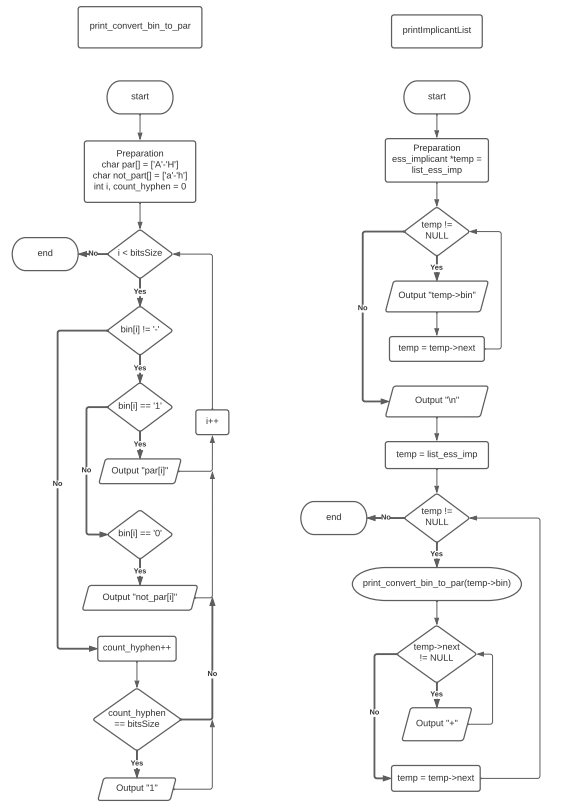
#### Gambar 7. *Flowchart* prosedur pairing

  
Gambar 8. Flowchart prosedur fillTable

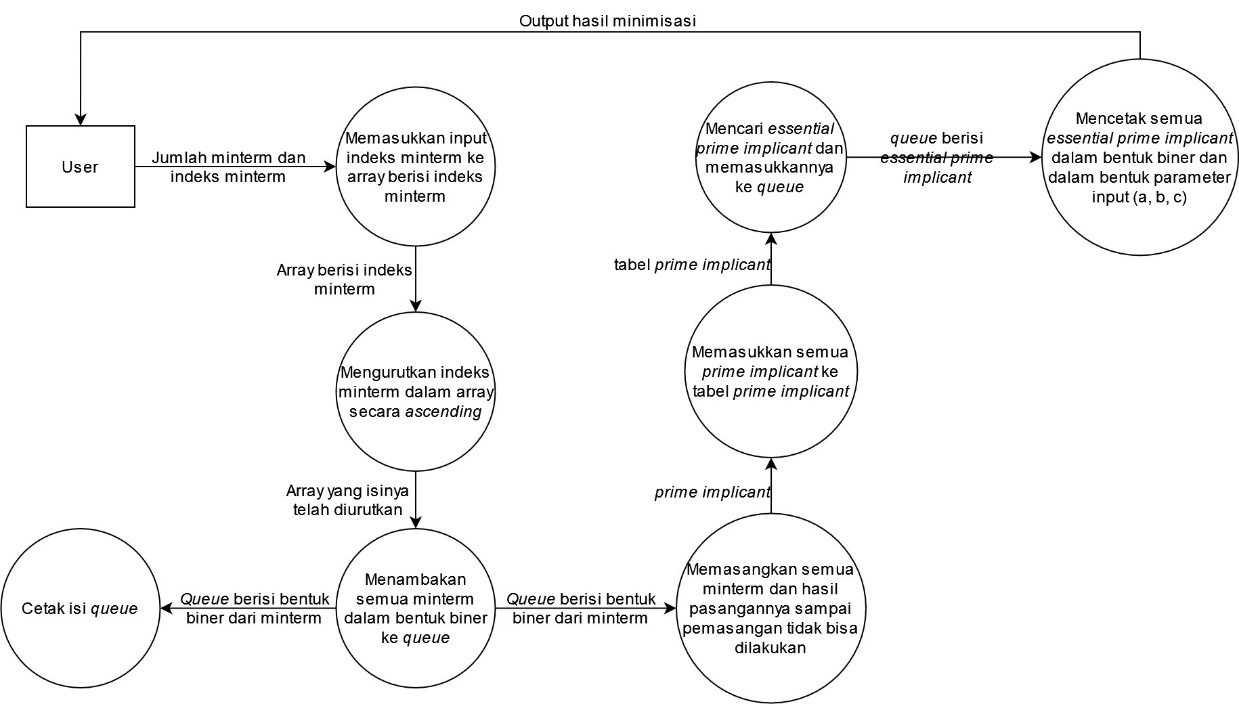
  
Gambar 9. Flowchart prosedur initTable, prosedur extract\_implicant, dan prosedur printTable

  
Gambar 10. Flowchart fungsi column\_check, fungsi isEmptyTable, dan prosedur addImplicant

  
Gambar 11. Flowchart fungsi is\_1\_implicant dan prosedur deleteRow\_oneimplicant

  
Gambar 12. Flowchart prosedur print\_convert\_bin\_to\_par dan prosedur printImplicantList

### *Data Flow Diagram*

  
Gambar 13. Data Flow Diagram program minimisasi

## **Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#define bitsSize 4

// Struct untuk implicant

**typedef** struct node

**{**

struct minterm **\***id**;**

char bin**[**bitsSize **+** 1**];**

int is\_paired**;**

struct node **\***next**;**

**}**Node**;**

// Struct untuk jumlah minterm setiap implicant

**typedef** struct minterm

**{**

int data**;**

struct minterm **\***next**;**

**}**Node\_minterm**;**

// Struct untuk prime implicant

**typedef** struct implicant

**{**

char bin**[**bitsSize **+** 1**];**

struct implicant **\***next**;**

**}**ess\_implicant**;**

// Queue untuk menampung implicant

**typedef** struct Queue

**{**

Node**\*** head**;**

**}**Queue**;**

void add**(**Node\_minterm **\*\***head**,** int dat**)**

// Prosedur menambah node implicant ke head

**{**

Node\_minterm**\*** new\_node **=** malloc**(sizeof(**Node\_minterm**));**

new\_node**->**data **=** dat**;**

new\_node**->**next **=** **NULL;**

**if(\***head **==** **NULL){**

**\***head **=** new\_node**;**

**}**

**else** **{**

Node\_minterm**\*** temp **=** **\***head**;**

**while(**temp**->**next **!=** **NULL){**

temp **=** temp**->**next**;**

**}**

temp**->**next **=** new\_node**;**

**}**

**}**

void printNode**(**Node\_minterm **\***head**)**

// Prosedur mencetak seluruh minterm dari sebuah implicant

**{**

Node\_minterm **\***temp **=** head**;**

**while(**temp **!=** **NULL){**

printf**(**"%d "**,** temp**->**data**);**

temp **=** temp**->**next**;**

**}**

**}**

int count\_minterm\_of\_implicant**(**Node\_minterm **\***head**)**

// Fungsi yang menghitung jumlah minterm setiap implicant

**{**

Node\_minterm **\***temp **=** head**;**

int count **=** 0**;**

**while(**temp **!=** **NULL){**

count**++;**

temp **=** temp**->**next**;**

**}**

**return** count**;**

**}**

void enqueue**(**Queue **\***queue**,** char inp**[],** Node\_minterm **\***id\_minterm**)**

// Prosedur menambah node baru dengan data (bentuk biner dari implicant dan mintermnya) ke queue

**{**

Node **\***head **=** queue**->**head**;**

Node**\*** new\_node **=** malloc**(sizeof(**Node**));**

new\_node**->**id **=** id\_minterm**;**

new\_node**->**is\_paired **=** 0**;**

strcpy**(**new\_node**->**bin**,** inp**);**

new\_node**->**next **=** **NULL;**

**if(**head **==** **NULL){**

queue**->**head **=** new\_node**;**

**}**

**else** **{**

Node**\*** temp **=** queue**->**head**;**

int is\_same\_data **=** 0**;**

**while(**temp**->**next **!=** **NULL** **&&** **!**is\_same\_data**){**

**if(!**strcmp**(**inp**,** temp**->**bin**)){**

is\_same\_data **=** 1**;**

**}**

temp **=** temp**->**next**;**

**}**

**if(!**is\_same\_data**){**

**if(**strcmp**(**inp**,** temp**->**bin**)){**

temp**->**next **=** new\_node**;**

**}**

**}**

**}**

**}**

void dequeue**(**Queue **\***queue**)**

// Prosedur menghapus node pada queue

**{**

int temp **=** 0**;**

Node**\*** head\_temp **=** queue**->**head**;**

**if(**head\_temp **!=** **NULL){**

**if(**head\_temp**->**next **!=** **NULL){**

queue**->**head **=** head\_temp**->**next**;**

**}**

**else** **{**

queue**->**head **=** **NULL;**

**}**

**}**

free**(**head\_temp**);**

**}**

void printQueue**(**Queue **\***queue**)**

// Prosedur mencetak isi queue

**{**

Node**\*** temp **=** queue**->**head**;**

**while(**temp **!=** **NULL){**

printf**(**"%s "**,** temp**->**bin**,** temp**->**id**->**data**);**

printNode**(**temp**->**id**);**

temp **=** temp**->**next**;**

printf**(**"\n"**);**

**}**

printf**(**"\n"**);**

**}**

int element\_in\_queue**(**Queue **\***queue**)**

// Fungsi yang mencetak jumlah node pada queue

**{**

Node**\*** temp **=** queue**->**head**;**

int dim **=** 0**;**

**while(**temp **!=** **NULL){**

dim**++;**

temp **=** temp**->**next**;**

**}**

**return** dim**;**

**}**

void subdestroy\_list**(**Node**\*** cur\_node**)**

// Fungsi untuk membebaskan semua node setelah head

**{**

**if(**cur\_node **!=** **NULL){**

subdestroy\_list**(**cur\_node**->**next**);**

free**(**cur\_node**);**

**}**

**}**

void destroy\_list**(**Node **\*\***head**)**

//Fungsi untuk membebaskan semua node selain head dan head itu sendiri

**{**

subdestroy\_list**(\***head**);**

**\***head **=** **NULL;**

**}**

void destroy\_implicant\_list**(**ess\_implicant **\***list\_implicant**)**

**{**

**if(**list\_implicant **!=** **NULL){**

destroy\_implicant\_list**(**list\_implicant**->**next**);**

free**(**list\_implicant**);**

**}**

**}**

void dec2bin**(**int bil**,** char hasil**[])**

// Prosedur yang mengubah indeks minterm ke bentuk binernya

**{**

**for(**int i **=** 0**;** i **<** bitsSize**;** **++**i**){**

**if(**bil **>=** pow**(**2**,** bitsSize **-** 1 **-** i**)){**

hasil**[**i**]** **=** '1'**;**

bil **-=** pow**(**2**,** bitsSize **-** 1 **-** i**);**

**}**

**else{**

hasil**[**i**]** **=** '0'**;**

**}**

**}**

**}**

int isPairingPossible**(**char arr1**[],** char arr2**[])**

// Fungsi yang mengecek apakah arr1 bisa dipasangkan dengan arr2

**{**

int dif\_element **=** 0**,** i**;**

**for(**i **=** 0**;** i **<** bitsSize**;** **++**i**){**

**if(**arr1**[**i**]** **!=** arr2**[**i**]){**

dif\_element**++;**

**}**

**}**

**if(**dif\_element **==** 1**){**

**return** 1**;**

**}**

**else{**

**return** 0**;**

**}**

**}**

int id\_dif**(**char arr1**[],** char arr2**[])**

// Fungsi yang menghasilkan indeks dari elemen kedua array yang berbeda

**{**

int id\_dif\_element **=** 0**;**

**while(**arr1**[**id\_dif\_element**]** **==** arr2**[**id\_dif\_element**]){**

id\_dif\_element**++;**

**}**

**return** id\_dif\_element**;**

**}**

void pairing**(**Queue **\***queue**,** Queue **\***next\_queue**)**

// Prosedur memasangkan minterm

**{**

Node **\***temp1 **=** queue**->**head**;**

Node **\***temp2**;**

int pairing\_done **=** 0**;**

**while(**temp1 **!=** **NULL){**

temp2 **=** temp1**->**next**;**

**while(**temp2 **!=** **NULL){**

**if(**isPairingPossible**(**temp1**->**bin**,** temp2**->**bin**)){**

pairing\_done**++;**

temp1**->**is\_paired **=** 1**;**

temp2**->**is\_paired **=** 1**;**

char mod\_bin**[**bitsSize **+** 1**];**

int id\_edit **=** id\_dif**(**temp1**->**bin**,** temp2**->**bin**);**

strcpy**(**mod\_bin**,** temp1**->**bin**);**

mod\_bin**[**id\_edit**]** **=** '-'**;**

Node\_minterm **\***temp\_minterm **=** **NULL;**

Node\_minterm **\***id\_temp1 **=** temp1**->**id**;**

Node\_minterm **\***id\_temp2 **=** temp2**->**id**;**

**while(**id\_temp1 **!=** **NULL){**

add**(&**temp\_minterm**,** id\_temp1**->**data**);**

id\_temp1 **=** id\_temp1**->**next**;**

**}**

**while(**id\_temp2 **!=** **NULL){**

add**(&**temp\_minterm**,** id\_temp2**->**data**);**

id\_temp2 **=** id\_temp2**->**next**;**

**}**

enqueue**(**next\_queue**,** mod\_bin**,** temp\_minterm**);**

**}**

temp2 **=** temp2**->**next**;**

**}**

temp1 **=** temp1**->**next**;**

**}**

**if(**pairing\_done **>** 0**){**

// Kasus ada minterm yang belum dipasangkan

temp1 **=** queue**->**head**;**

**while(**temp1 **!=** **NULL){**

**if(**temp1**->**is\_paired **==** 0**){**

enqueue**(**next\_queue**,** temp1**->**bin**,** temp1**->**id**);**

**}**

temp1 **=** temp1**->**next**;**

**}**

**}**

**}**

void initTable**(**int brs**,** int kol**,** int table**[**brs**][**kol**])**

// Prosedur untuk inisiasi tabel dengan nilai nol

**{**

int i**,** j**;**

**for(**i **=** 0**;** i **<** brs**;** **++**i**){**

**for(**j **=** 0**;** j **<** kol**;** **++**j**){**

table**[**i**][**j**]** **=** 0**;**

**}**

**}**

**}**

void extract\_implicant**(**Queue **\***queue**,** char arr**[][**bitsSize **+** 1**],** int minterm\_per\_implicant**[])**

// Prosedur memasukkan semua implicant ke arr dan memasukkan jumlah minterm setiap implicant ke array minterm\_per\_implicant

**{**

int i **=** 0**;**

Node **\***temp **=** queue**->**head**;**

**while(**temp **!=** **NULL){**

strcpy**(**arr**[**i**],** temp**->**bin**);**

minterm\_per\_implicant**[**i**]** **=** count\_minterm\_of\_implicant**(**temp**->**id**);**

i**++;**

temp **=** temp**->**next**;**

**}**

**}**

void fillTable**(**Queue **\***queue**,** int arr\_minterm**[],** int brs**,** int kol**,** int table**[**brs**][**kol**])**

// Prosedur mengisi tabel essential prime implicant

**{**

int i**,** j**;**

Node **\***temp\_queue **=** queue**->**head**;**

Node\_minterm **\***temp\_list\_minterm **=** temp\_queue**->**id**;**

**for(**i **=** 0**;** i **<** brs**;** **++**i**){**

**for(**j **=** 0**;** j **<** kol**;** **++**j**){**

**if(**temp\_list\_minterm **!=** **NULL){**

**if(**temp\_list\_minterm**->**data **==** arr\_minterm**[**j**]){**

table**[**i**][**j**]** **=** 1**;**

temp\_list\_minterm **=** temp\_list\_minterm**->**next**;**

**}**

**}**

**}**

**if(**temp\_queue**->**next **!=** **NULL){**

temp\_queue **=** temp\_queue**->**next**;**

temp\_list\_minterm **=** temp\_queue**->**id**;**

**}**

**}**

**}**

void printTable**(**int brs**,** int kol**,** int table**[**brs**][**kol**])**

// Prosedur mencetak tabel essential prime implicant

**{**

int i**,** j**;**

**for(**i **=** 0**;** i **<** brs**;** **++**i**){**

**for(**j **=** 0**;** j **<** kol**;** **++**j**){**

printf**(**"%d "**,** table**[**i**][**j**]);**

**}**

printf**(**"\n"**);**

**}**

**}**

int column\_check**(**int brs**,** int kol**,** int kol\_check**,** int table**[**brs**][**kol**],** int **\***id\_brs**)**

/\* Fungsi untuk mengecek apakah suatu kolom hanya terdiri dari satu elemen bernilai 1

Pointer id\_brs digunakan untuk menyimpan indeks baris/prime implicant dari elemen bernilai 1 tersebut \*/

**{**

int count **=** 0**,** id **=** 0**,** i**,** j**;**

**while(**count **!=** 2 **&&** id **<** brs**){**

**if(**table**[**id**][**kol\_check**]** **==** 1**){**

count**++;**

**\***id\_brs **=** id**;**

**}**

id**++;**

**}**

**if(**count **==** 1**){**

**return** 1**;**

**}**

**else{**

**return** 0**;**

**}**

**}**

int isEmptyTable**(**int brs**,** int kol**,** int table**[**brs**][**kol**])**

// Fungsi untuk mengecek apakah tabel kosong

**{**

int i **=** 0**,** j**,** isFound **=** 0**;**

**while(!**isFound **&&** i **<** brs**){**

j **=** 0**;**

**while(**j **<** kol **&&** **!**isFound**){**

**if(**table**[**i**][**j**]** **==** 1**){**

isFound **=** 1**;**

**}**

**else{**

j**++;**

**}**

**}**

i**++;**

**}**

**if(**isFound**){**

**return** 0**;**

**}**

**else{**

**return** 1**;**

**}**

**}**

void addImplicant**(**ess\_implicant **\*\***list\_ess\_implicant**,** char implicant\_bin**[])**

// Prosedur menambah prime implicant ke node list\_ess\_implicant

**{**

ess\_implicant **\***new\_impli **=** malloc**(sizeof(**ess\_implicant**));**

strcpy**(**new\_impli**->**bin**,** implicant\_bin**);**

new\_impli**->**next **=** **NULL;**

**if(\***list\_ess\_implicant **==** **NULL){**

**\***list\_ess\_implicant **=** new\_impli**;**

**}**

**else{**

ess\_implicant **\***temp **=** **\***list\_ess\_implicant**;**

**while(**temp**->**next **!=** **NULL){**

temp **=** temp**->**next**;**

**}**

temp**->**next **=** new\_impli**;**

**}**

**}**

int is\_1\_implicant**(**int brs**,** int kol**,** int brs\_check**,** int table**[**brs**][**kol**],** int **\***id\_minterm**)**

/\* Fungsi untuk mengecek apakah suatu baris memiliki satu elemen bernilai 1

Pointer id\_minterm digunakan untuk menyimpan indeks minterm dari elemen bernilai 1 tersebut \*/

**{**

int count **=** 0**,** id\_kol **=** 0**;**

**while(**id\_kol **<** kol**){**

**if(**table**[**brs\_check**][**id\_kol**]** **==** 1**){**

count**++;**

**\***id\_minterm **=** id\_kol**;**

**}**

id\_kol**++;**

**}**

**if(**count **==** 1**){**

**return** 1**;**

**}**

**else{**

**return** 0**;**

**}**

**}**

void deleteRow\_oneimplicant**(**int brs**,** int kol**,** int table**[**brs**][**kol**],** int minterm\_per\_implicant**[])**

// Prosedur menghapus/mengosongkan baris pada tabel prime implicant

**{**

int i**,** j**;**

int temp\_id\_minterm **=** 0**,** temp\_unused **=** 0**;**

**for(**i **=** 0**;** i **<** brs**;** **++**i**){**

**if(**is\_1\_implicant**(**brs**,** kol**,** i**,** table**,** **&**temp\_id\_minterm**)** **&&** minterm\_per\_implicant**[**i**]** **!=** 1**){**

**if(!**column\_check**(**brs**,** kol**,** temp\_id\_minterm**,** table**,** **&**temp\_unused**)){**

**for(**j **=** 0**;** j **<** kol**;** **++**j**){**

table**[**i**][**j**]** **=** 0**;**

**}**

**}**

**}**

**}**

**}**

void print\_convert\_bin\_to\_par**(**char bin**[])**

// Prosedur mencetak hasil minimisasi dalam bentuk parameter input

**{**

char par**[]** **=** **{**'A'**,** 'B'**,** 'C'**,** 'D'**,** 'E'**,** 'F' **,** 'G'**,** 'H'**};**

char not\_par**[]** **=** **{**'a'**,** 'b'**,** 'c'**,** 'd'**,** 'e'**,** 'f'**,** 'g'**,** 'h'**};**

int i**,** count\_hyphen **=** 0**;**

**for(**i **=** 0**;** i **<** bitsSize**;** **++**i**){**

**if(**bin**[**i**]** **!=** '-'**){**

**if(**bin**[**i**]** **==** '1'**){**

printf**(**"%c"**,** par**[**i**]);**

**}**

**else** **if(**bin**[**i**]** **==** '0'**){**

printf**(**"%c"**,** not\_par**[**i**]);**

**}**

**}**

**else{**

count\_hyphen**++;**

**if(**count\_hyphen **==** bitsSize**){**

printf**(**"1"**);**

**}**

**}**

**}**

**}**

void printImplicantList**(**ess\_implicant **\***list\_ess\_imp**)**

// Prosedur mencetak hasil minimisasi dalam bentuk biner dan diikuti oleh bentuk parameter input

**{**

ess\_implicant **\***temp **=** list\_ess\_imp**;**

**while(**temp **!=** **NULL){**

printf**(**"%s "**,** temp**->**bin**);**

temp **=** temp**->**next**;**

**}**

printf**(**"\n"**);**

temp **=** list\_ess\_imp**;**

**while(**temp **!=** **NULL){**

print\_convert\_bin\_to\_par**(**temp**->**bin**);**

**if(**temp**->**next **!=** **NULL){**

printf**(**" + "**);**

**}**

temp **=** temp**->**next**;**

**}**

**}**

int main**(){**

// Deklarasi variabel

Queue **\***queue **=** malloc**(sizeof(**queue**));**

Queue **\***next\_queue **=** malloc**(sizeof(**queue**));**

ess\_implicant **\***list\_ess\_imp **=** **NULL;**

queue**->**head **=** **NULL;**

next\_queue**->**head **=** **NULL;**

int jml\_mint**,** id\_mint**,** i**,** j**,** n\_iter **=** 1**,** temp\_sort**;**

char arr\_inp**[**bitsSize **+** 1**];**

arr\_inp**[**bitsSize**]** **=** '\0'**;**

// Menerima input jumlah minterm

printf**(**"Input : "**);**

scanf**(**"%d"**,** **&**jml\_mint**);**

**if(**jml\_mint **<** 1 **||** jml\_mint **>** pow**(**2**,** bitsSize**)){**

printf**(**"Input jumlah minterm tidak valid!"**);**

free**(**queue**);**

free**(**next\_queue**);**

exit**(**1**);**

**}**

int arr\_minterm**[**jml\_mint**];**

// Menerima input indeks minterm

**for(**i **=** 0**;** i **<** jml\_mint**;** **++**i**){**

printf**(**"Minterm ke : "**);**

scanf**(**"%d"**,** **&**arr\_minterm**[**i**]);**

**if(**arr\_minterm**[**i**]** **<** 0**){**

printf**(**"\n!Warning! Karena tidak ada minterm negatif, maka dipilih minterm 0\n\n"**);**

arr\_minterm**[**i**]** **=** 0**;**

**}**

**else** **if(**arr\_minterm**[**i**]** **>** pow**(**2**,** bitsSize**)** **-** 1**){**

printf**(**"\n!Warning! Karena indeks minterm melebihi minterm maksimal, maka dipilih minterm maksimal sesuai ukuran bit\n\n"**);**

arr\_minterm**[**i**]** **=** pow**(**2**,** bitsSize**)** **-** 1**;**

**}**

**}**

// Sort indeks minterm

**for(**i **=** 0**;** i **<** jml\_mint**;** **++**i**){**

**for(**j **=** 0**;** j **<** jml\_mint **-** 1**;** **++**j**){**

**if(**arr\_minterm**[**j**]** **>** arr\_minterm**[**j **+** 1**]){**

temp\_sort **=** arr\_minterm**[**j **+** 1**];**

arr\_minterm**[**j **+** 1**]** **=** arr\_minterm**[**j**];**

arr\_minterm**[**j**]** **=** temp\_sort**;**

**}**

**}**

**}**

// Konversi minterm ke biner dan masukkan ke queue

**for(**i **=** 0**;** i **<** jml\_mint**;** **++**i**){**

dec2bin**(**arr\_minterm**[**i**],** arr\_inp**);**

Node\_minterm **\***temp **=** **NULL;**

add**(&**temp**,** arr\_minterm**[**i**]);**

enqueue**(**queue**,** arr\_inp**,** temp**);**

**}**

// Cetak list minterm input

printf**(**"\nList minterm input :\n"**);**

printQueue**(**queue**);**

// Melakukan algoritma Quine-McCluskey sekali

pairing**(**queue**,** next\_queue**);**

// Melakukan algoritma Quine-McCluskey sampai tidak bisa dilakukan lagi

**while(**next\_queue**->**head **!=** **NULL){**

printf**(**"Iterasi %d :\n"**,** n\_iter**);**

n\_iter**++;**

printQueue**(**next\_queue**);**

queue**->**head **=** next\_queue**->**head**;**

next\_queue**->**head **=** **NULL;**

pairing**(**queue**,** next\_queue**);**

**}**

// Deklarasi variabel untuk tabel essential prime implicant

int n\_imp **=** element\_in\_queue**(**queue**);**

int EprimeImp**[**n\_imp**][**jml\_mint**];**

char list\_implicant**[**n\_imp**][**bitsSize **+** 1**];**

int minterm\_hold\_by\_implicant**[**n\_imp**];**

// Inisiasi tabel

initTable**(**n\_imp**,** jml\_mint**,** EprimeImp**);**

// Memasukkan semua prime implicant ke list\_implicant dan jumlah minterm setiap implicant ke minterm\_hold\_by\_implicant

extract\_implicant**(**queue**,** list\_implicant**,** minterm\_hold\_by\_implicant**);**

// Mengisi tabel essential prime implicant

fillTable**(**queue**,** arr\_minterm**,** n\_imp**,** jml\_mint**,** EprimeImp**);**

// Mencari essential prime implicant sampai tabel kosong

**while(!**isEmptyTable**(**n\_imp**,** jml\_mint**,** EprimeImp**)){**

id\_mint **=** 0**;**

int id\_implicant**;**

**while(!**column\_check**(**n\_imp**,** jml\_mint**,** id\_mint**,** EprimeImp**,** **&**id\_implicant**)** **&&** id\_mint **<** jml\_mint**){**

id\_mint**++;**

**}**

**if(**id\_mint **!=** jml\_mint**){**

addImplicant**(&**list\_ess\_imp**,** list\_implicant**[**id\_implicant**]);**

**for(**i **=** 0**;** i **<** jml\_mint**;** **++**i**){**

**if(**EprimeImp**[**id\_implicant**][**i**]** **==** 1**){**

**for(**j **=** 0**;** j **<** n\_imp**;** **++**j**){**

EprimeImp**[**j**][**i**]** **=** 0**;**

**}**

EprimeImp**[**id\_implicant**][**i**]** **=** 0**;**

**}**

**}**

**}**

deleteRow\_oneimplicant**(**n\_imp**,** jml\_mint**,** EprimeImp**,** minterm\_hold\_by\_implicant**);**

**}**

// Mencetak essential prime implicant

printf**(**"Hasil minimisasi (not A = a) : \n"**);**

printImplicantList**(**list\_ess\_imp**);**

destroy\_list**(&(**queue**->**head**));**

destroy\_list**(&(**next\_queue**->**head**));**

destroy\_implicant\_list**(**list\_ess\_imp**);**

free**(**queue**);**

free**(**next\_queue**);**

**return** 0**;**

**}**

# **Kesimpulan dan Lesson Learned**

* Minimisasi merupakan sebuah proses penyederhanaan sebuah fungsi boolean yang sangat bermanfaat, karena dapat mengurangi *cost* serta kompleksitas dari sebuah rangkaian
* Quine-McCluskey merupakan salah satu metode minimisasi rangkaian yang mudah untuk diimplementasikan ke dalam suatu program, karena di dalamnya menggunakan sistem eliminasi yang berulang, sehingga dapat diselesaikan secara rekursif. Selain itu metode ini juga lebih efektif untuk menyederhanakan suatu rangkaian yang memiliki banyak variabel.
* Deklarasi array sebagai isi suatu *structure* dengan panjang yang bergantung dengan nilai variabel di luar *struct* tidak bisa dilakukan, kecuali dengan *header define*

# **Pembagian Tugas dalam Kelompok**

|  |  |
| --- | --- |
| Nama Anggota Kelompok | Kontribusi |
| Adro Anra Purnama | Membuat flowchart fungsi dan prosedur |
| Surya Dharma | Membuat *source code* program, *data flow diagram*, flowchart algoritma |
| Fariz Iftikhar Falakh | Melengkapi laporan, membuat file presentasi |
| Senggani Fatah Sedayu | Melengkapi laporan |

# **Link Repository**

<https://github.com/DentAlpha/Minimisasi-Quine-McCluskey>

# **Referensi**

1. Brown S. D. dan Vranesic Z. G, *Fundamentals of digital logic with VHDL design*, 2000.