TUGAS BESAR *LOGIC MINIMIZATION*

EL2008 PEMECAHAN MASALAH DENGAN C

Kelompok 2

Anggota:

Eraraya Morenzo Muten / 18320003

Michelle Angelina / 18320007

Shadrina Syahla Vidyana / 18320031

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2022

DAFTAR ISI

DESKRIPSI SIMULASI *LOGIC MINIMIZATION*

Deskripsi dan Latar Belakang Permasalahan

Permasalahan yang diangkat untuk topik Tugas Besar 2022 adalah permasalahan *logic minimization*. Minimisasi logika merupakan suatu proses menyederhanakan suatu fungsi aljabar *Boolean*. Setiap fungsi *Boolean* dinyatakan dalam bentuk *sum of minterms* atau *product of maxterms*. Menyederhanakan suatu fungsi ekspresi aljabar *Boolean* dapat menggunakan beberapa cara, yaitu metode manipulasi aljabar atau K-Maps. Penurunan metode K-Maps untuk menyelesaikan fungsi ekspresi aljabar *Boolean* yang lebih besar dinamakan metode Quine-McCluskey (Metode tabulasi). Beberapa literatur merekomendasikan penggunaan metode Quine-McCluskey karena dianggap paling baik dalam penyederhanaan dan dapat digunakan untuk variable fungsi ekspresi aljabar yang besar. Metode tabulasi menggunakan tahap-tahap penyederhanaan yang jelas dan teratur dalam penyederhanaan suatu fungsi aljabar.

Perkembangan teknologi masa kini diharapkan dapat bekerja secara efisien dan cepat, tetapi juga benar dan simple untuk dirancang. Untuk mencapai hal tersebu, minimisasi logika fungsi aljabar *Boolean* diperlukan untuk mengurangi penggunaan kompleksitas sirkuit supaya menghasilkan program atau sirkuit yang efisien untuk digunakan. Selain itu, penyederhanaan fungsi aljabar *Boolean* juga dapat mengurangi biaya dan penggunaan *logic gate* pada suatu sirkuit atau rangkaian tanpa mengubah hasil keluaran rangkaian tersebut. Penyederhanaan logika *Boolean* juga dapat mengurangi kerusakan pada sirkuit *logic gate* karena mengurangi beban kerja *chip* pada rangkaian tersebut. Jika rangkaian *logic gate* pada kehidupan sehari-hari tidak diminimisasi, atau dengan kata lain dioptimalisasi, barang-barang digital yang digunakan saat ini tidak akan secepat itu dan lebih mahal karena komponen-komponennya yang tidak disederhanakan.

Deskripsi Proses Input dan Output

Input dari proses penyederhanaan fungsi aljabar *Boolean* adalah suatu fungsi aljabar *Boolean* dalam bentuk *truth table* dan memiliki kompleksitas tinggi. *Truth table* tersebut akan melalui proses penyederhanaan menggunakan metode Quine McCluskey. Masukan berupa *truth table* akan ditranslasi terlebih dahulu menjadi *minterm* sebagai elemen penting untuk menerapkan metode minimisasi Quine-McCluskey. Program akan dijalankan menggunakan bahasa pemrograman C. Input berupa fungsi aljabar *Boolean* yang tidak sederhana tersebut dalam bentuk *truth table* akan menghasilkan output fungsi aljabar ekspresi *Boolean* sederhana.

Berikut adalah langkah-langkah menyederhanakan fungsi aljabar *Boolean* menggunakan metode Quine McCluskey (metode tabulasi) sebagai solusi dari permasalahan:

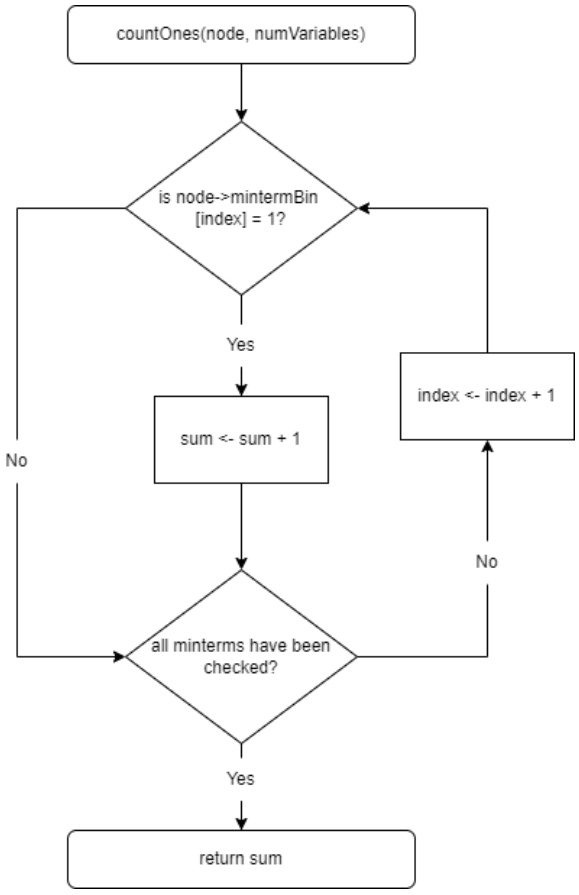
1. Menyusun suku-suku *minterms* berdasarkan binernya. Urutan harus selalu meningkat berdasarkan jumlah yang ada dalam ekuivalen biner. Dengan kata lain, dalam angka decimal, urutannya harus membesar.
2. Membandingkan suku *minterms* yang ada dalam grup yang berurutan. Jika ada perubahan hanya pada posisi satu bit, maka ambil pasangan dari dua suku *minterms* tersebut. Beri simbol “-“ di posisi bit yang berbeda dan pertahankan bit yang tersis apa adanya.

3. Mengulangi langkah 2 sampai suku-suku yang terbentuk berdasarkan kelompok tersebut sampai mendapatkan semua implikan prima.
4. Merumuskan table implikan prima tersebut. Menempatkan ‘1’ dalam sel yang sesuai dengan istilah *minterms* yang tercakup dalam setiap implikan prima.
5. Menemukan implikan prima esensial dengan memperhatikan setiap kolom. Jika suku *minterms* hanya dicakup oleh satu implikan prima (implikan prima esensial). Implikasi prima esensial tersebut akan menjadi bagian dari fungsi *Boolean* yang sederhana.
6. Mengurangi table implikan prima dengan menghilangkan baris dari setiap implikan prima esensial dan kolom yang sesuai dengan suku *minterms* yang tercakup dalam implikan prima esensial tersebut. Mengulangi langkah 5 sampai proses selesai saat fungsi *Boolean* tidak dapat disederhanakan lebih lanjut.

FLOWCHART

Flowchart Fungsi countOnes()

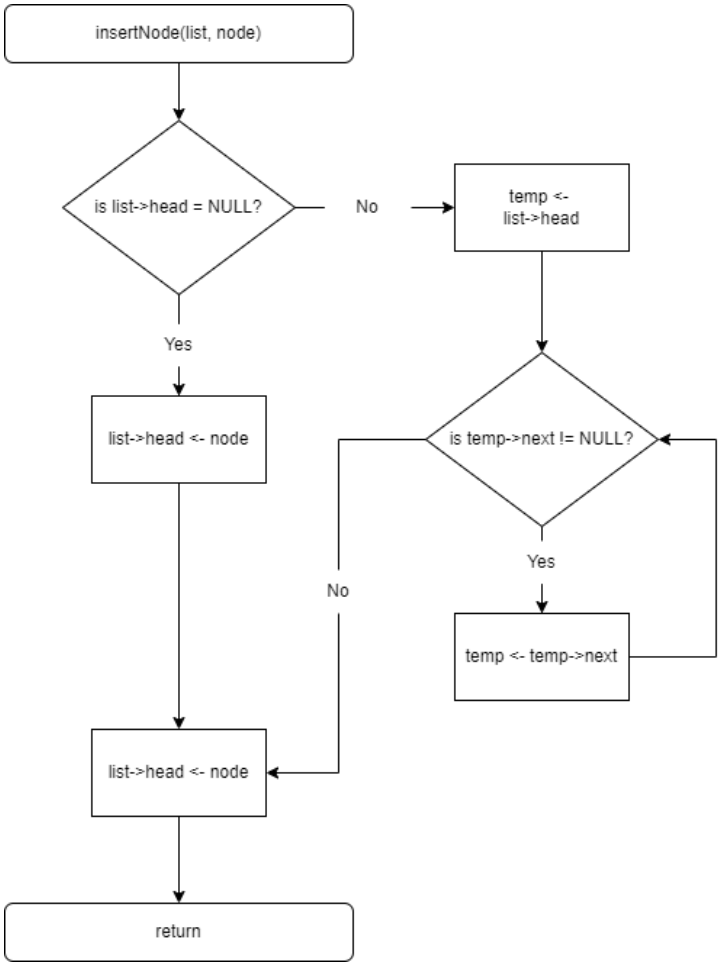
Fungsi ini digunakan untuk menghitung jumlah 1 pada *minterms*. Fungsi ini memanfaatkan *looping* yang memeriksa setiap data dalam *node minterm* apakah bernilai ‘1’. Jika ya, suatu variable bernama “sum” akan bertambah satu. *Loop* akan terus berjalan sampai semua *minterm* telah diperiksa dan “sum” akan menjadi nilai yang dikembalikan dari fungsi.



Gambar 1 CountOnes()

Flowchart Fungsi insertNode()

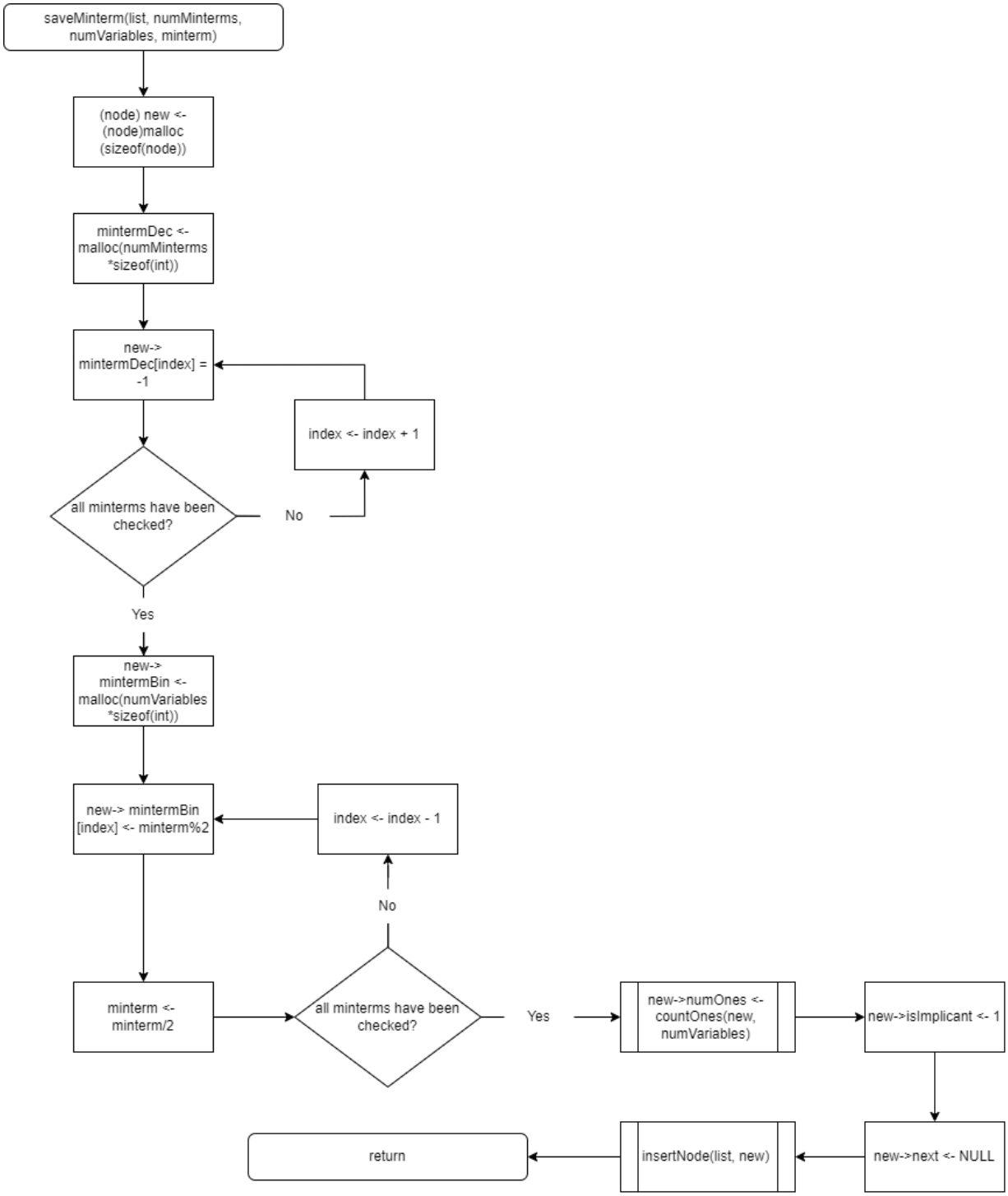
Fungsi ini digunakan untuk memasukkan *node* pada *struct*. Fungsi ini memanfaatkan *linked list*. Proses penyimpanan *node* memanfaatkan *looping* sampai ujung mulai dari *head list* sesuai dengan aturan penyimpanan ke dalam *linked list*. Jika *head list* masih kosong, *node* akan menjadi *head list* tersebut. Jika sudah terisi, *node* akan masuk ke *next*. Fungsi ini lebih tepatnya disebut sebagai prosedur dan tidak mengembalikan data apapun, prosedur merupakan proses yang akan dialami data yang dipanggil.



Gambar 2 InsertNode()

Flowchart Fungsi saveMinterm()

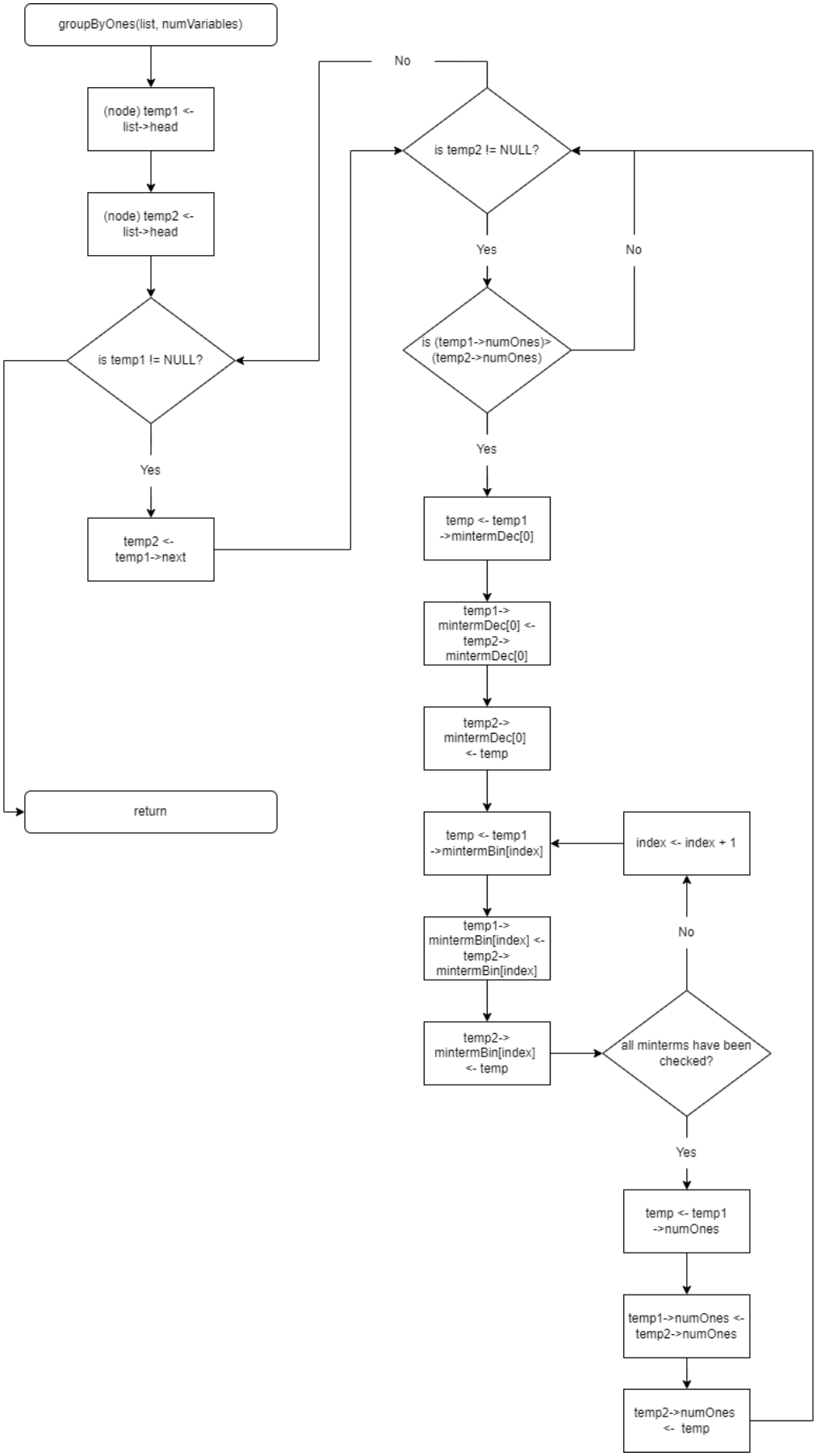
Fungsi ini digunakan untuk menyimpan *minterm* dalam bentuk biner dan memasukkan dalam bentuk *node*. Fungsi ini memanfaatkan memori yang dinamis untuk mengoptimalkan penggunaan memori. Penyimpanan biner dari *minterm* tersebut akan disimpan dalam bentuk *node*. Konversi angka decimal menjadi biner menggunakan operasi matematika modulo dan pembagian oleh 2 karena biner berbasis 2. Fungsi ini memanggil fungsi countOnes dan insertNode. Fungsi ini merupakan prosedur yang tidak mengembalikan suatu data.



Gambar 3 SaveMinterm()

Flowchart Fungsi groupByOnes()

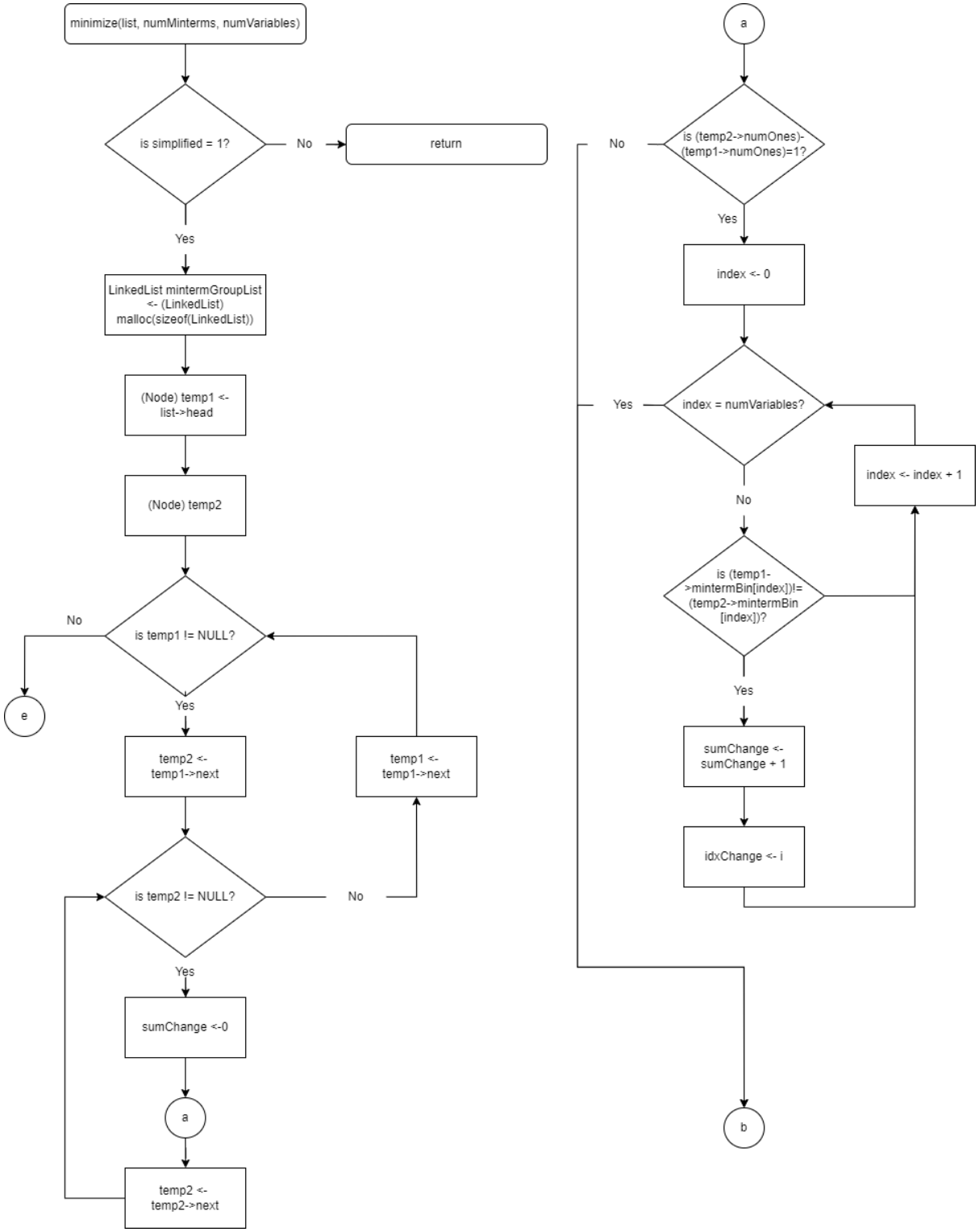
Fungsi ini digunakan untuk pengelompokkan *minterm* berdasarkan ‘1’. Fungsi akan menyusuri *linked list* yang telah disimpan. Fungsi akan membandingkan *node-node* kemudian menukar data tersebut. Penukaran data ini dilakukan untuk *minterm* dalam decimal dan juga biner. Semua data akan ditukar di dalam *list* tersebut. Setelah selesai melakukan penukaran, fungsi akan pindah ke *node* selanjutnya (*next*).

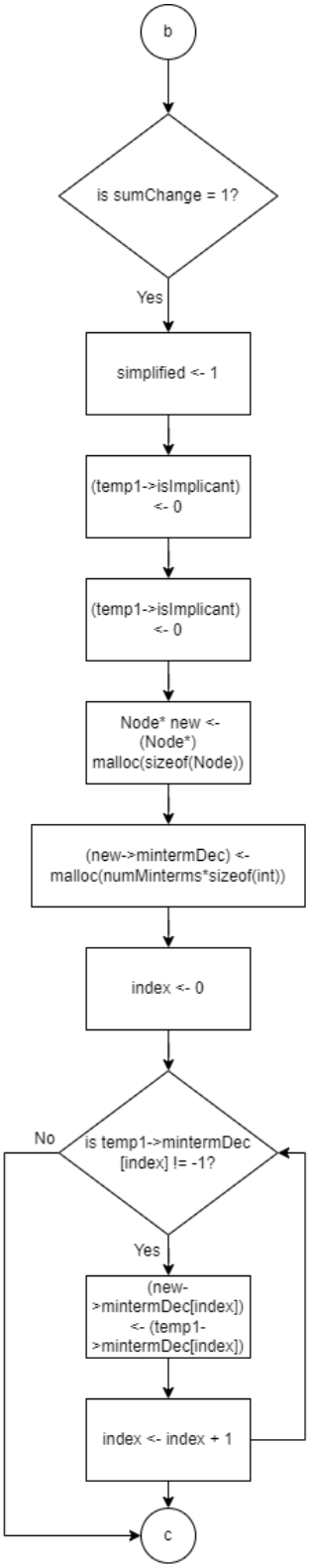


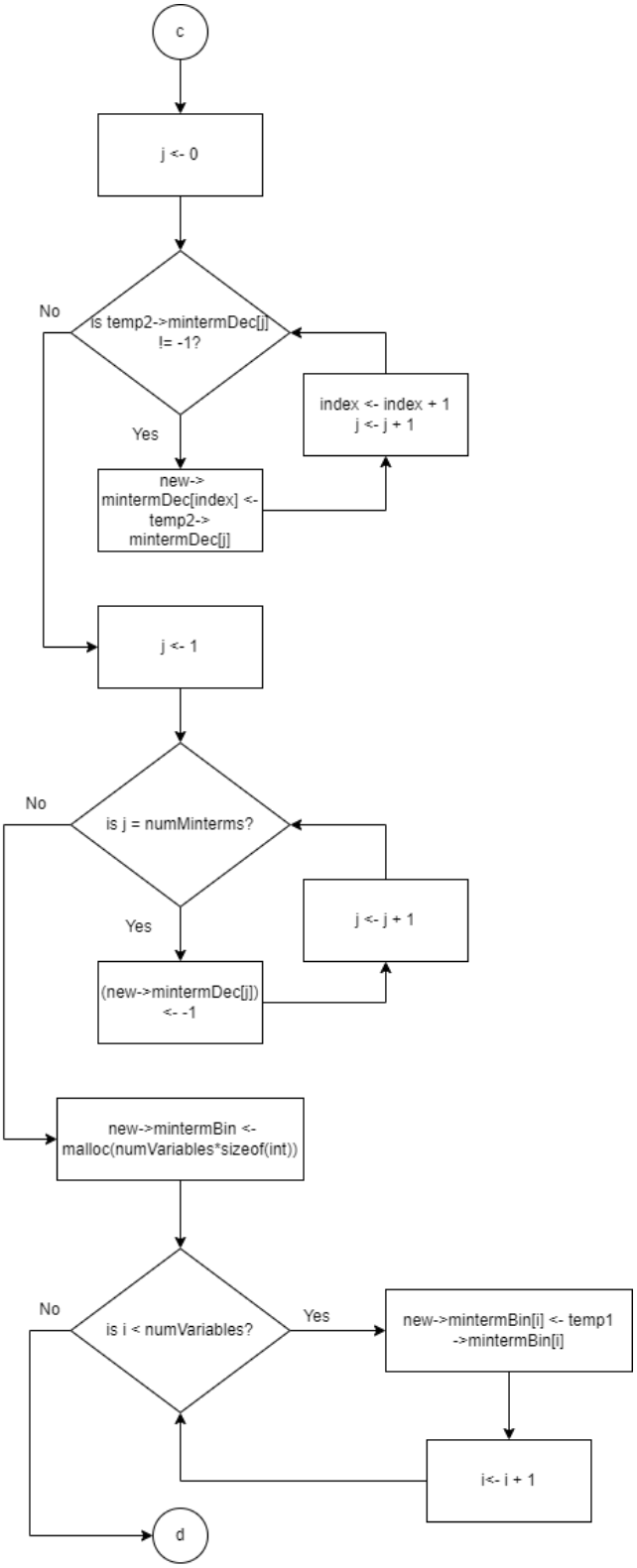
Gambar 4 GroupByOnes()

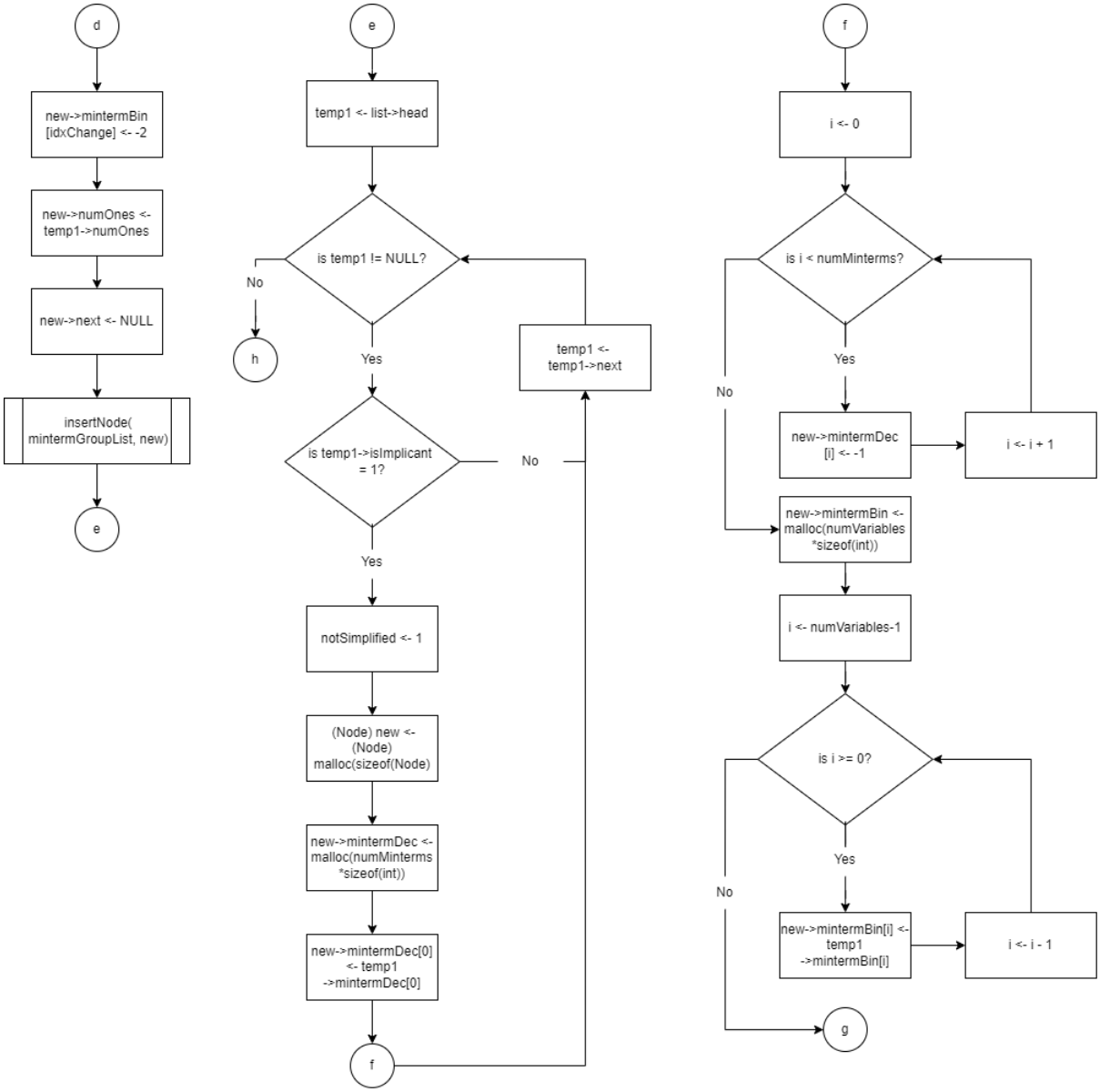
Flowchart Fungsi minimize()

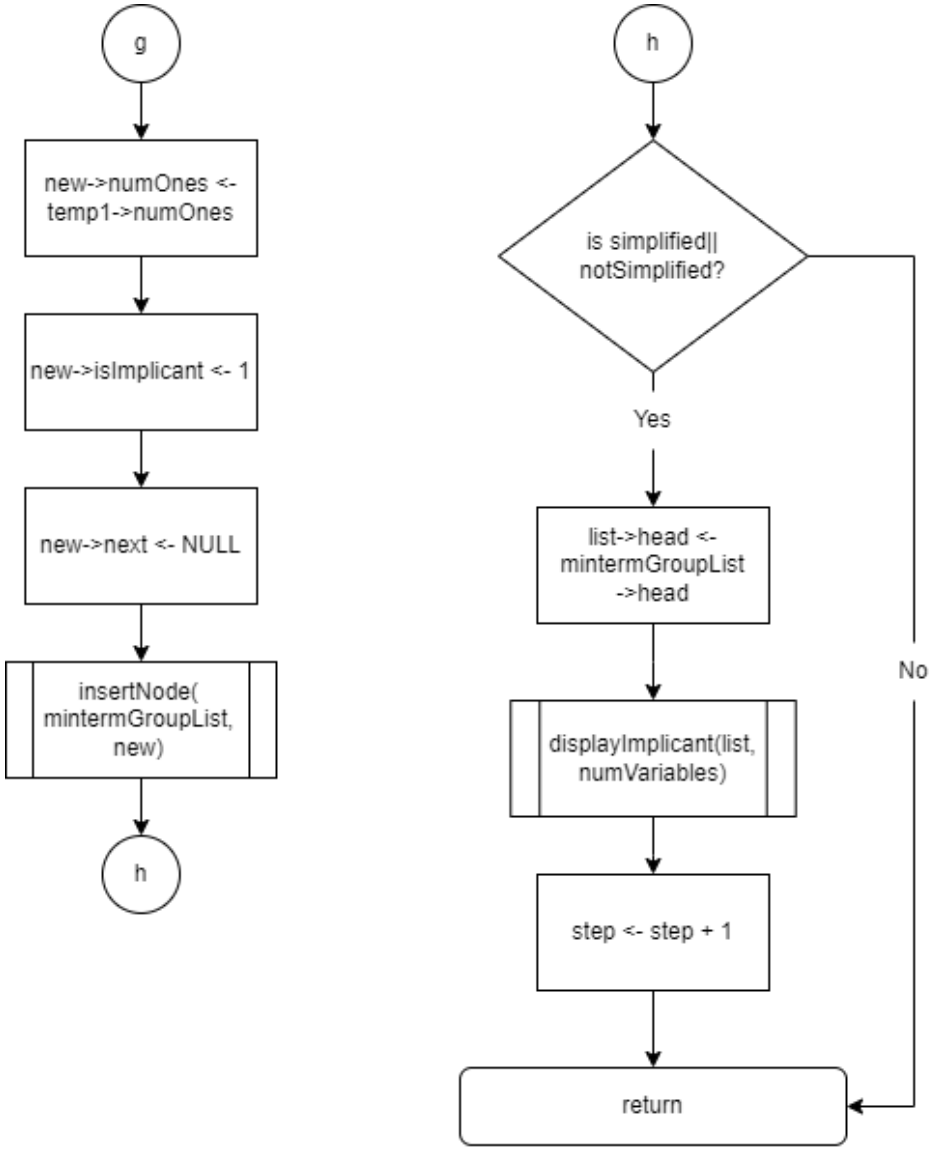
Fungsi ini digunakan untuk implementasi proses Quine McCluskey untuk menyederhanakan fungsi. Fungsi menggunakan variabel “simplified” sebagai tanda keberadaan implikan yang disederhanakan. Prosedur *minimize* ini akan membuat *linked list* penyimpanan *minterm* yang telah disederhanakan. Kemudian fungsi akan menyusuri *linked list* dan juga melakukan pembandingan *node*. Setelah itu akan ada perbandingan kelompok *minterm* yang telah dikelompokkan sesuai dengan prosedur metode Quine-McCluskey. Jika ada suatu perubahan dalam kelompok tersebut, variabel “sumChange” bernilai ‘1’ dan akan ditandai adanya penyederhanaan pada “simplified”. Indikator implikan *minterm* pada *node* akan diubah. *Node* akan ditambahkan ke dalam *list* kemudian ada pengisian data baru. *Array minterm* akan diisi dengan -1. Bit yang berubah akan ditandai, dan lanjut ke *node* berikutnya. Untuk *minterm* yang tidak tersederhanakan, *minterm* tersebut merupakan implikan sehingga tidak dapat disederhanakan. Prosedur tersebut akan menghasilkan *linked list* berisi hasil penyederhanaan *minterm*.


minimize(list, numMinterms, numVariables)
is simplified = 1?
No
return
Yes
LinkedList mintermGroupList <- (LinkedList) malloc(sizeof(LinkedList))
(Node) temp1 <- list->head
(Node) temp2
is temp1 != NULL?
No
e
Yes
temp2 <- temp1->next
temp1 <- temp1->next
is temp2 != NULL?
No
Yes
sumChange <-0
a
temp2 <- temp2->next
a
is (temp2->numOnes)-(temp1->numOnes)=1?
No
Yes
index <- 0
index = numVariables?
Yes
No
index <- index + 1
is (temp1->mintermBin[index])!=(temp2->mintermBin[index])?
Yes
sumChange <- sumChange + 1
idxChange <- i
b


b
is sumChange = 1?
Yes
simplified <- 1
(temp1->isImplicant) <- 0
(temp1->isImplicant) <- 0
Node* new <- (Node*) malloc(sizeof(Node))
(new->mintermDec) <- malloc(numMinterms*sizeof(int))
index <- 0
No
is temp1->mintermDec [index] != -1?
Yes
(new->mintermDec[index]) <- (temp1->mintermDec[index])
index <- index + 1
c


c
j <- 0
No
is temp2->mintermDec[j] != -1?
Yes
new-> mintermDec[index] <- temp2-> mintermDec[j]
index <- index + 1
j <- j + 1
j <- 1
No
is j = numMinterms?
Yes
(new->mintermDec[j]) <- -1
j <- j + 1
new->mintermBin <- malloc(numVariables*sizeof(int))
No
is i < numVariables?
Yes
new->mintermBin[i] <- temp1 ->mintermBin[i]
i<- i + 1
d

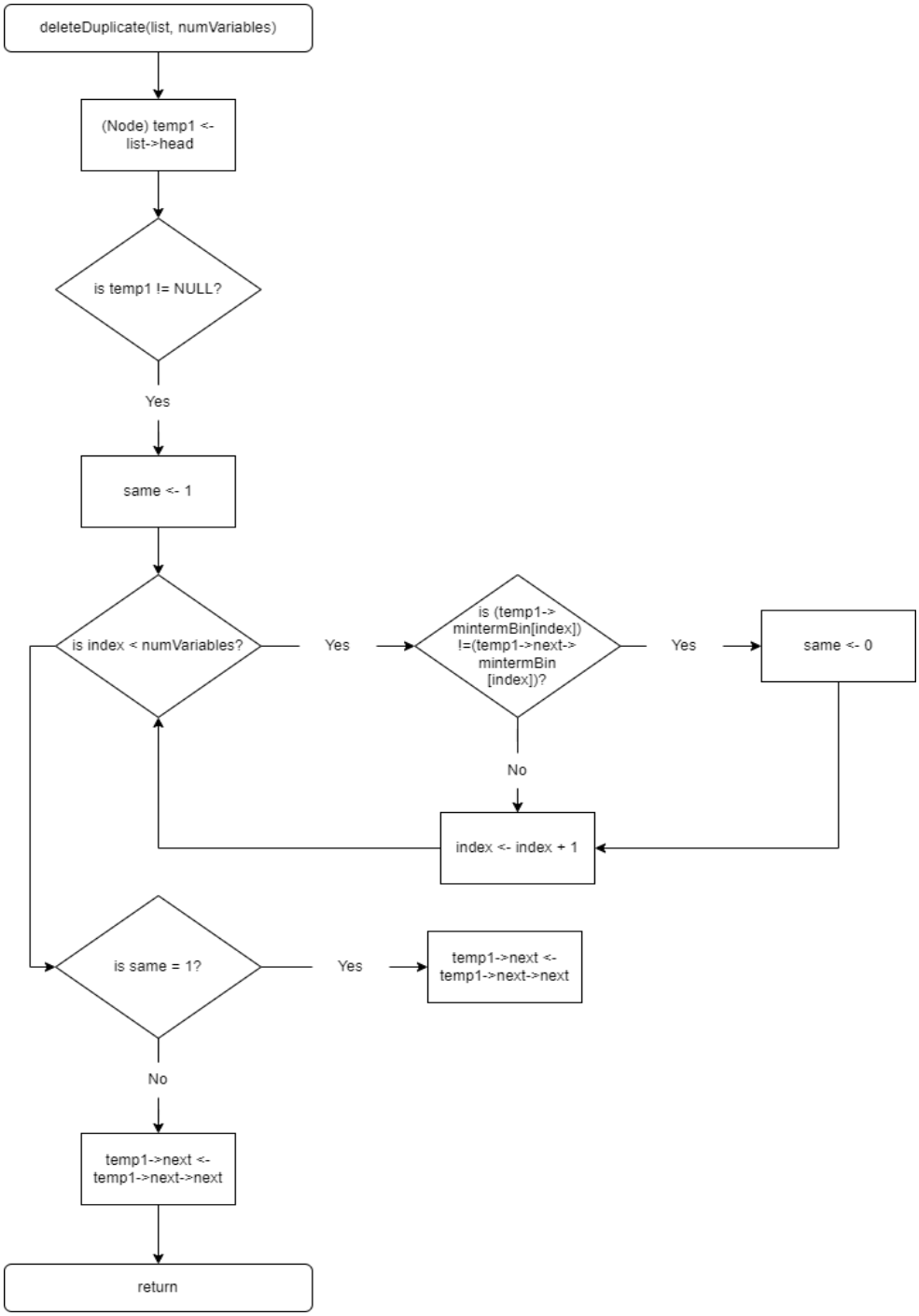

d
new->mintermBin [idxChange] <- -2
new->numOnes <- temp1->numOnes
new->next <- NULL
insertNode(mintermGroupList, new)
e
e
temp1 <- list->head
is temp1 != NULL?
No
h
Yes
is temp1->isImplicant = 1?
No
Yes
notSimplified <- 1
(Node) new <- (Node) malloc(sizeof(Node)
new->mintermDec <- malloc(numMinterms *sizeof(int))
new->mintermDec[0] <- temp1 ->mintermDec[0]
f
temp1 <- temp1->next
f
i <- 0
is i < numMinterms?
Yes
No
new->mintermDec [i] <- -1
i <- i + 1
new->mintermBin <- malloc(numVariables *sizeof(int))
i <- numVariables-1
is i >= 0?
Yes
No
new->mintermBin[i] <- temp1 ->mintermBin[i]
i <- i - 1
g



Gambar 5 Minimize()

Flowchart Fungsi deleteDuplicate()

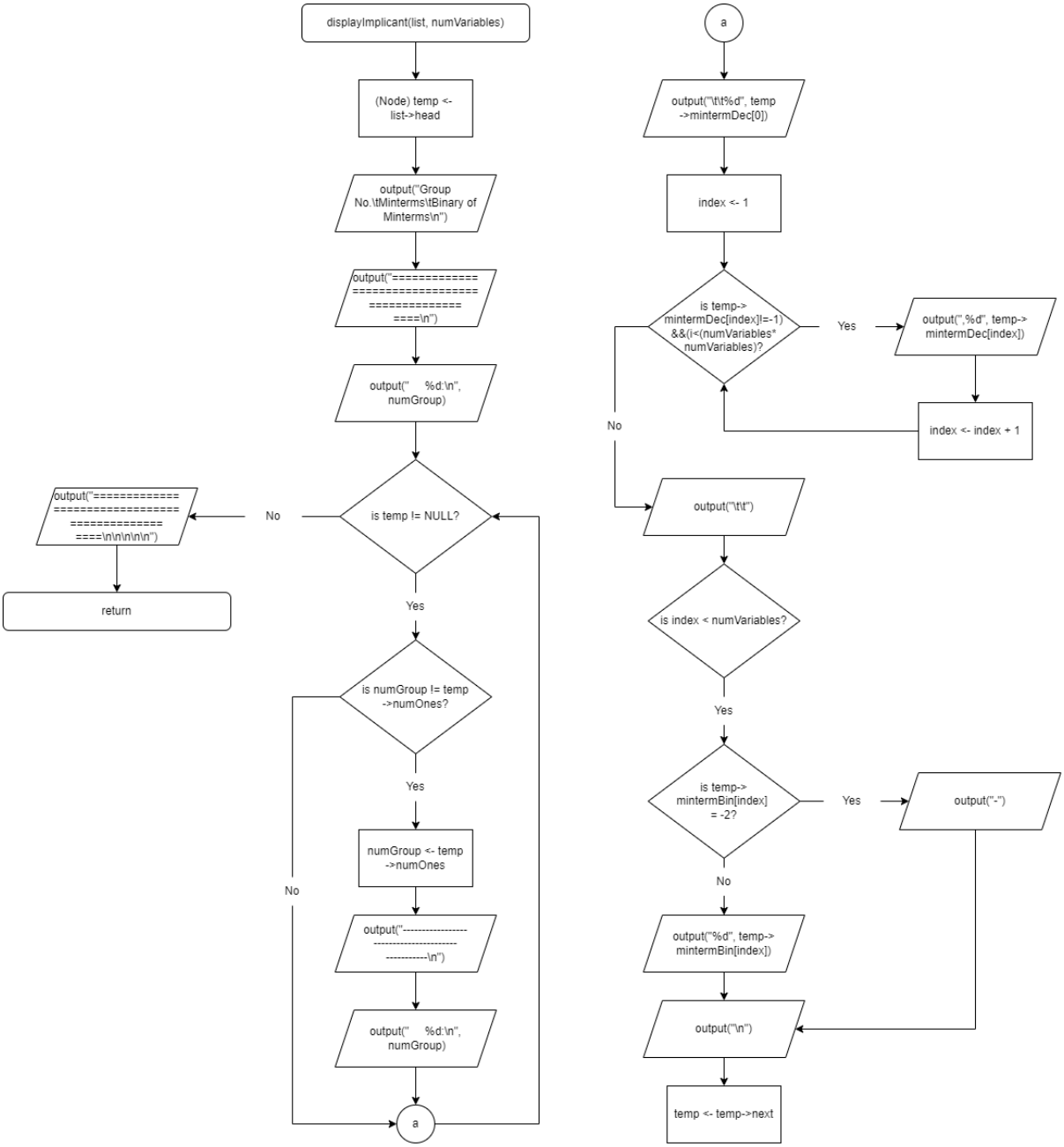
Fungsi ini digunakan untuk menghapus *minterm* yang sama dalam table tabulasi. Hasil *grouping* akan mengakibatkan hasil kelompok yang terduplikat karena pengurutan *minterm* yang berbeda. Oleh karena itu, suatu prosedur yang menghapus kelompok yang terduplikat tersebut. Pada “mintermBin” akan ditelusuri, jika menemukan yang sama akan dihilangkan. Pemeriksaan dilakukan per bit, jika ada yang tidak sama, fungsi akan langsung mengubah tanda variabel “same” menjadi ‘0’. Jika keduanya sama, implikan akan dilewati.



Gambar 6 deleteDuplicate()

Flowchart Fungsi displayImplicant()

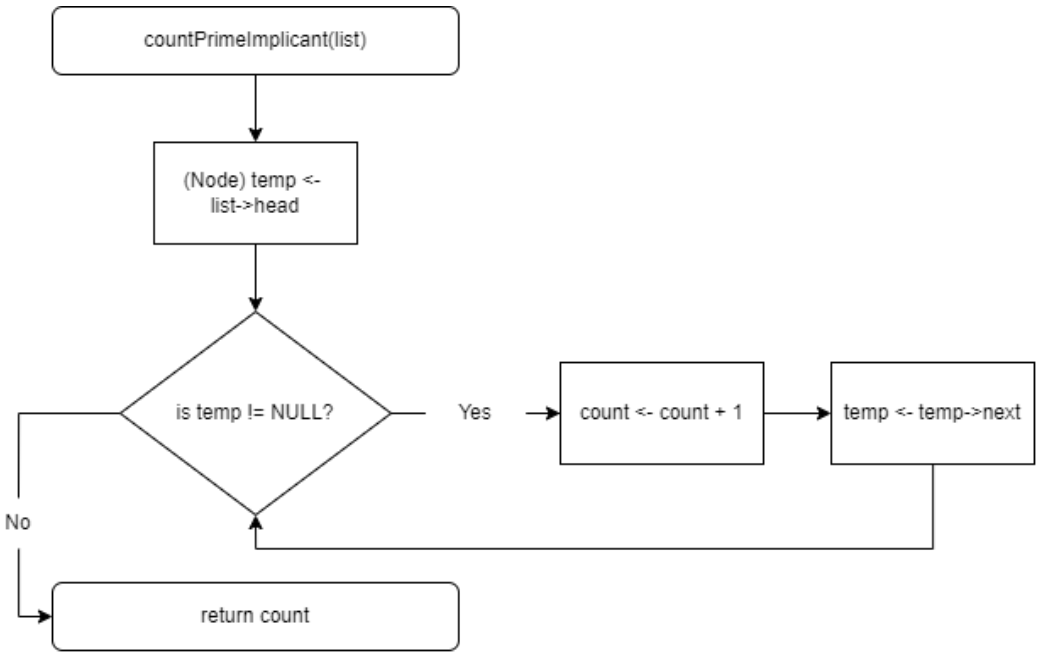
Fungsi ini digunakan untuk menampilkan proses *minimize* atau implementasi proses Quine-McCluskey ke layar untuk menunjukkan proses tabulasi fungsi aljabar ekspresi *boolean*. Prosedur ini akan mencetak judul tabel kemudian menyusuri *list* untuk mencari implikan. Program akan mencetak implikan tersebut dalam bentuk biner. Kemudian, fungsi akan lanjut ke *node* berikutnya.



Gambar 7 DisplayImplicant()

Flowchart Fungsi countPrimeImplicant()

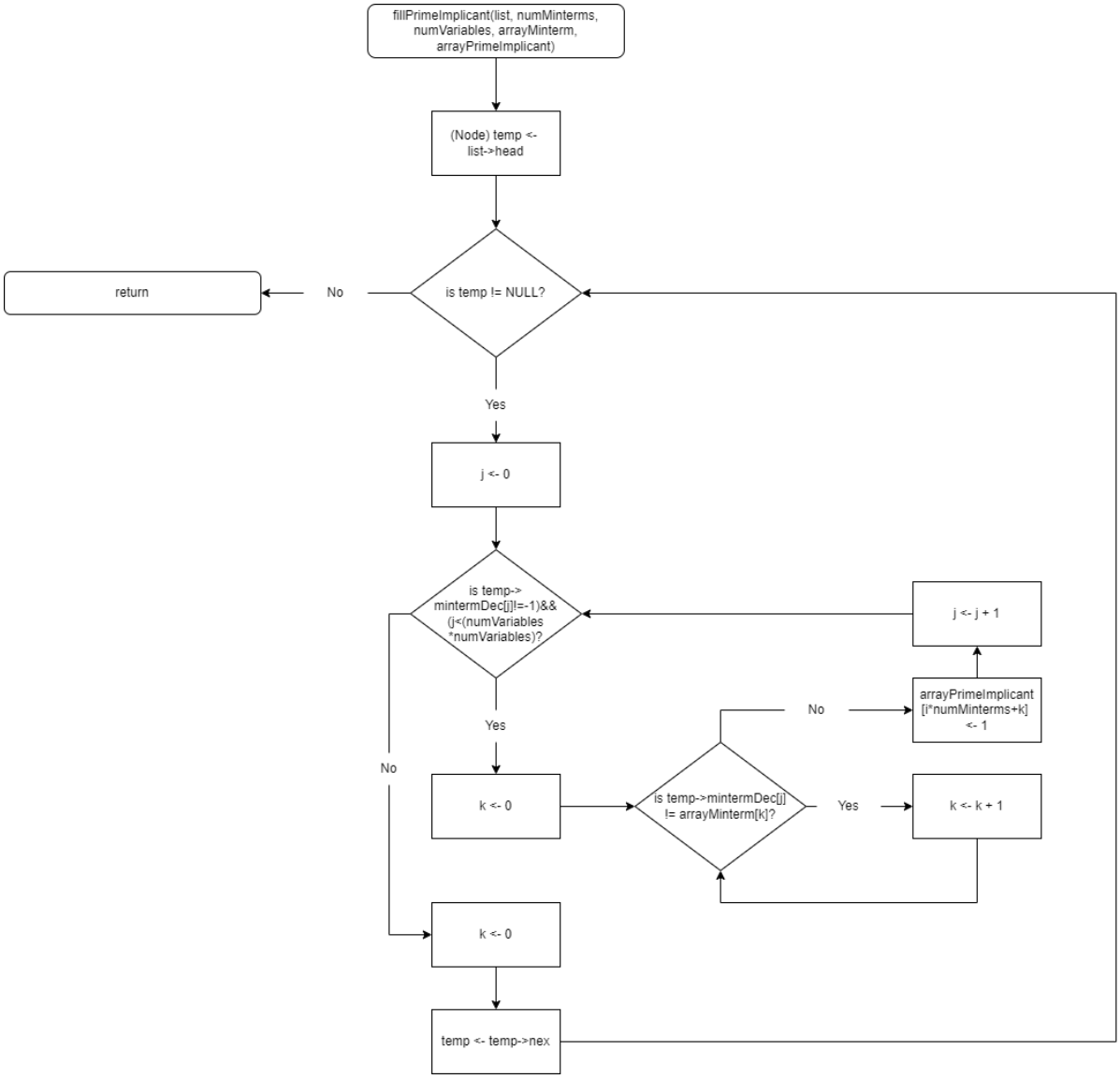
Fungsi untuk menghitung jumlah *prime implicant* yang terbentuk. Fungsi akan menyusuri kembali *linked list* kemudian jika menemukan implikan akan menambah jumlah variabel "count" sebanyak satu. Kemudian fungsi akan lanjut ke *node* berikutnya dan mengembalikan variabel "count".



Gambar 8 CountPrimeImplicant()

Flowchart Fungsi fillPrimeImplicant()

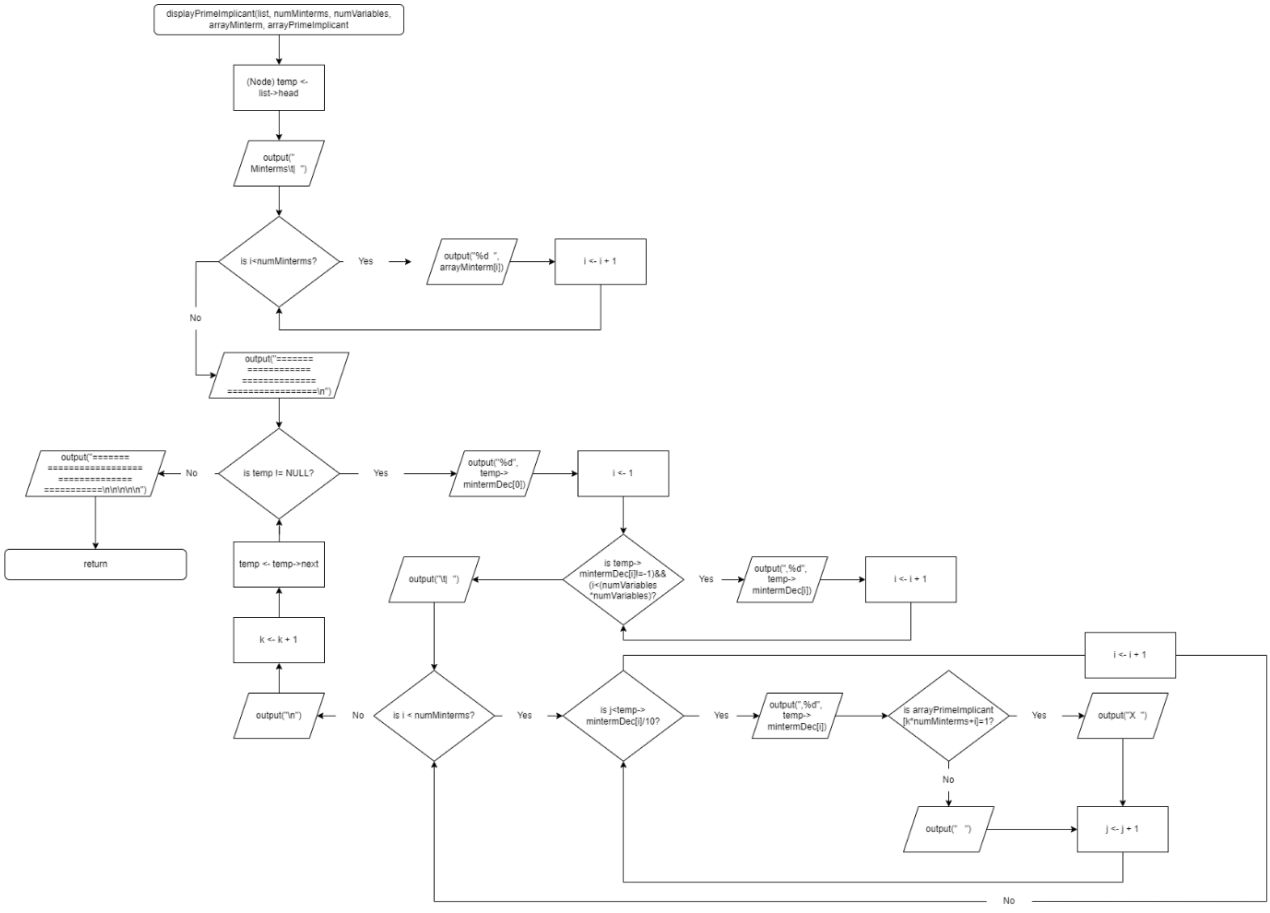
Fungsi ini digunakan untuk menyimpan *prime implicant*. Prosedur ini akan mengisi matriks *prime implicant* dengan memanfaatkan *looping* supaya semua *node* dan *list* terpanggil.



Gambar 9 FillPrimeImplicant()

Flowchart Fungsi displayPrimeImplicant()

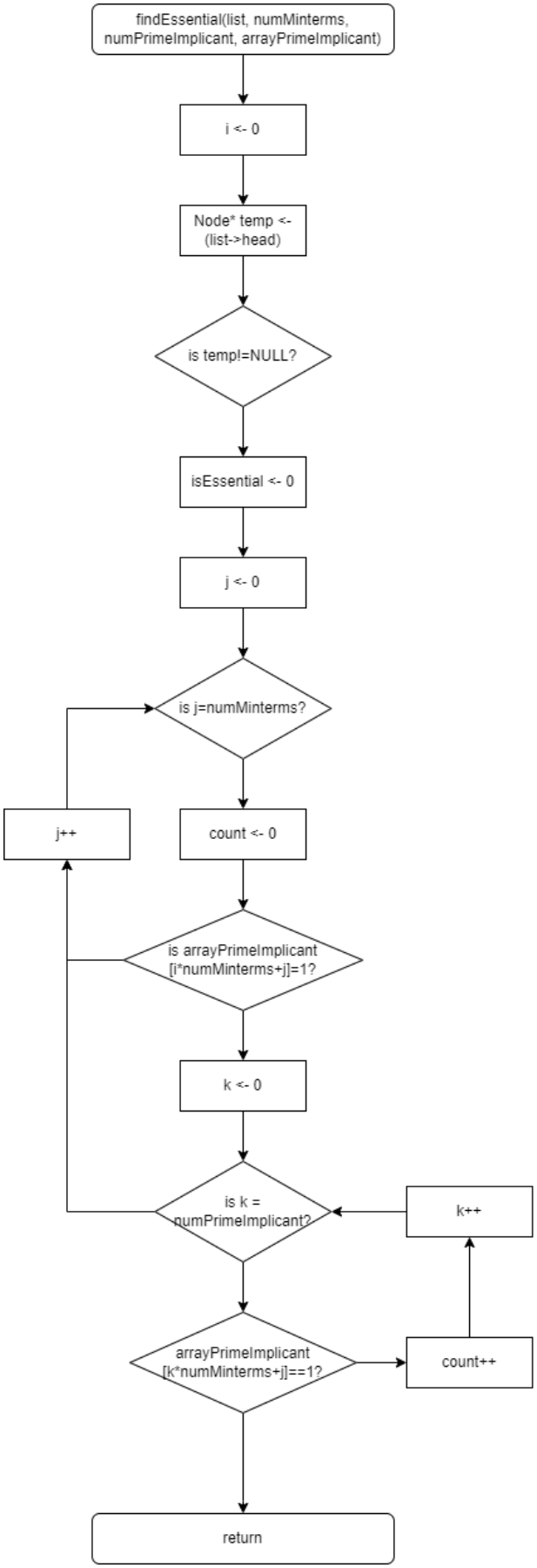
Fungsi ini digunakan untuk menampilkan *prime implicant*. Prosedur ini akan mencetak judul tabel kemudian menyusuri *list* untuk mencari implikan. Program akan mencetak implikan prima *minterm* yang bersangkutan. Kemudian, fungsi akan lanjut ke *node* berikutnya.



Gambar 10 DisplayPrimeImplicant()

Flowchart Fungsi findEssential()

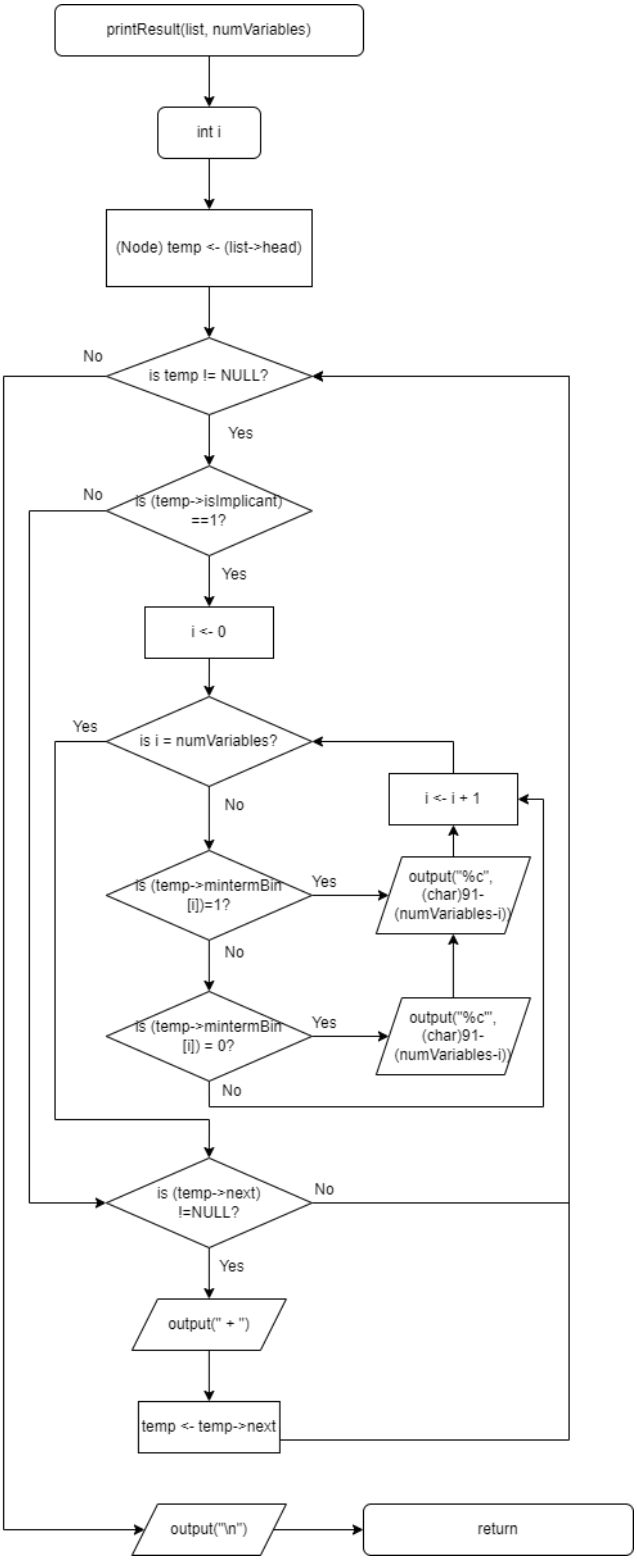
Fungsi ini digunakan untuk mencari implikan prima yang bersifat esensial. Fungsi ini berperan penting untuk menyederhanakan fungsi aljabar *Boolean*. Fungsi akan Kembali menyusuri *linked list* kemudian akan memeriksa baris yang ada. Jika terdapat data yang berisi ‘1’, akan dilakukan pemeriksaan kolom tabel. Jika terdapat kolom yang berisi hanya satu buah bit ‘1’, *prime implicant* tersebut dapat dikatakan sebagai esensial. Kemudian prosedur ini akan berlanjut ke *node* berikutnya.



Gambar 11 FindEssential()

Flowchart Fungsi printResult()

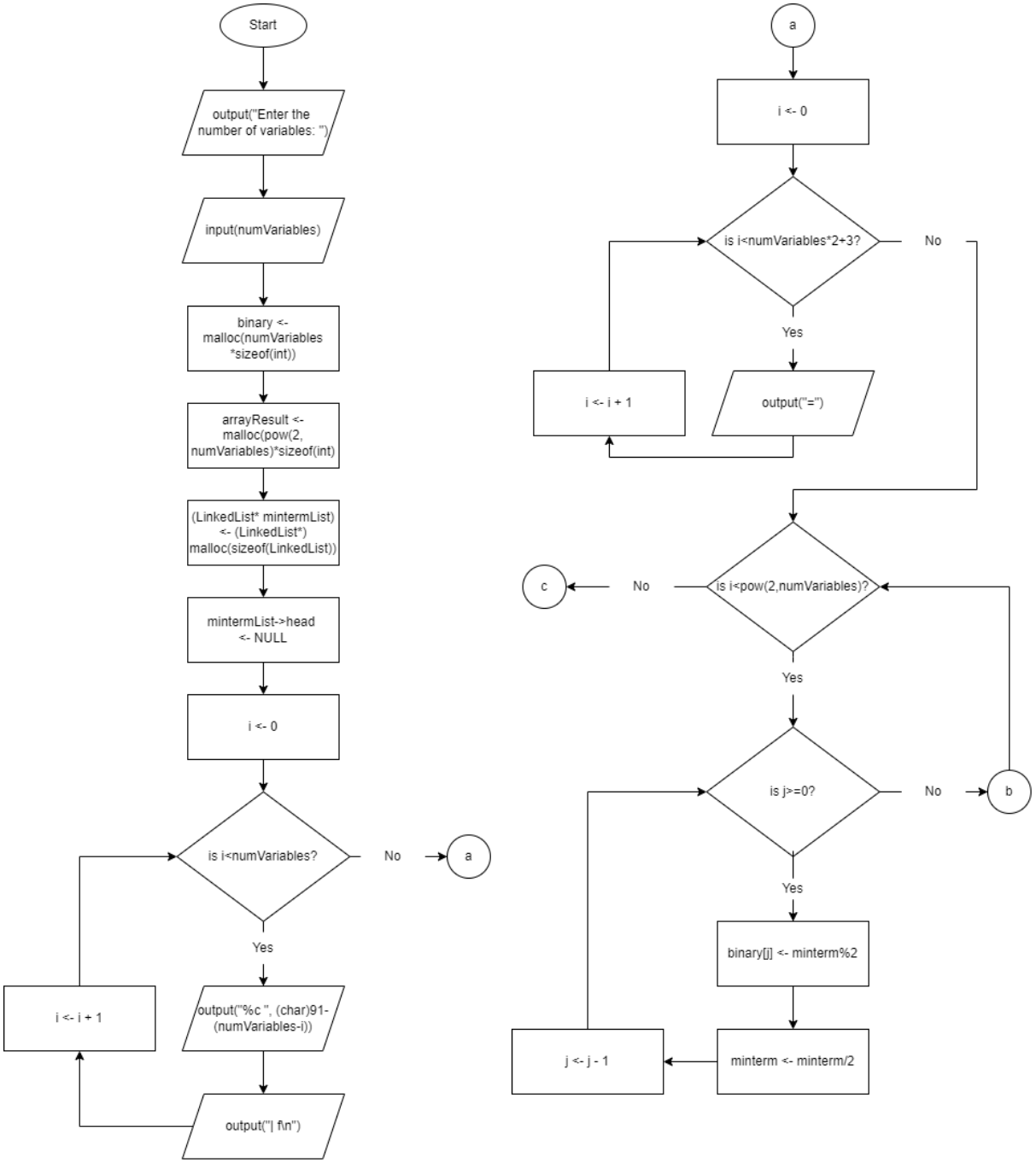
Fungsi yang digunakan untuk keperluan mencetak hasil penyederhanaan fungsi aljabar *Boolean*. Prosedur ini akan mencari *node-node* yang implikan prima bersifat esensial untuk dicetak kepada pengguna. Setelah itu, program akan bergeser ke *node* selanjutnya.

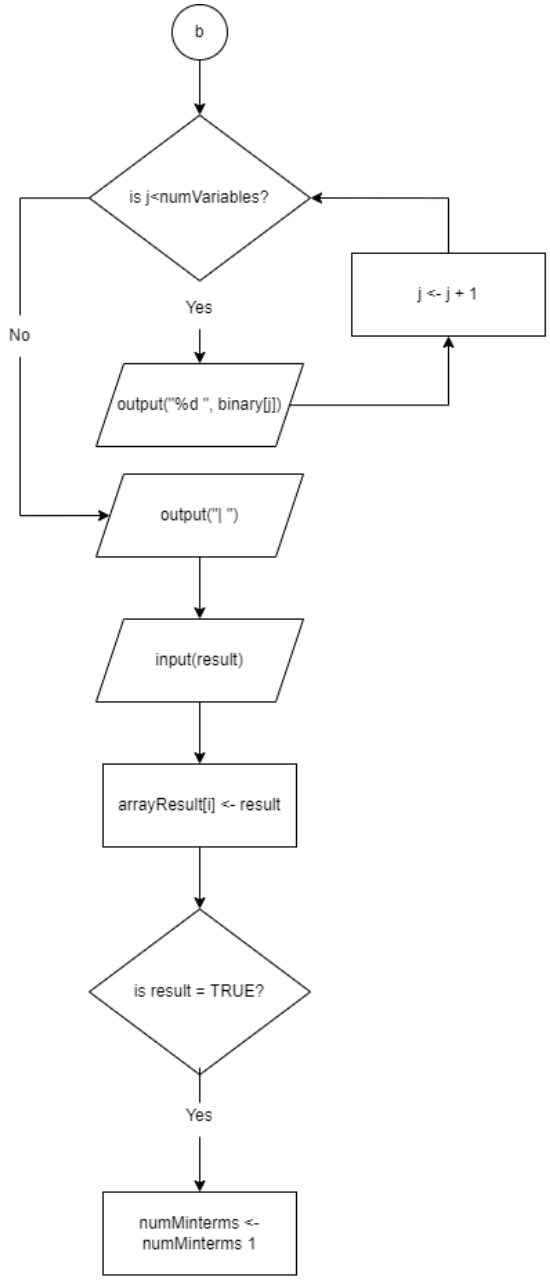


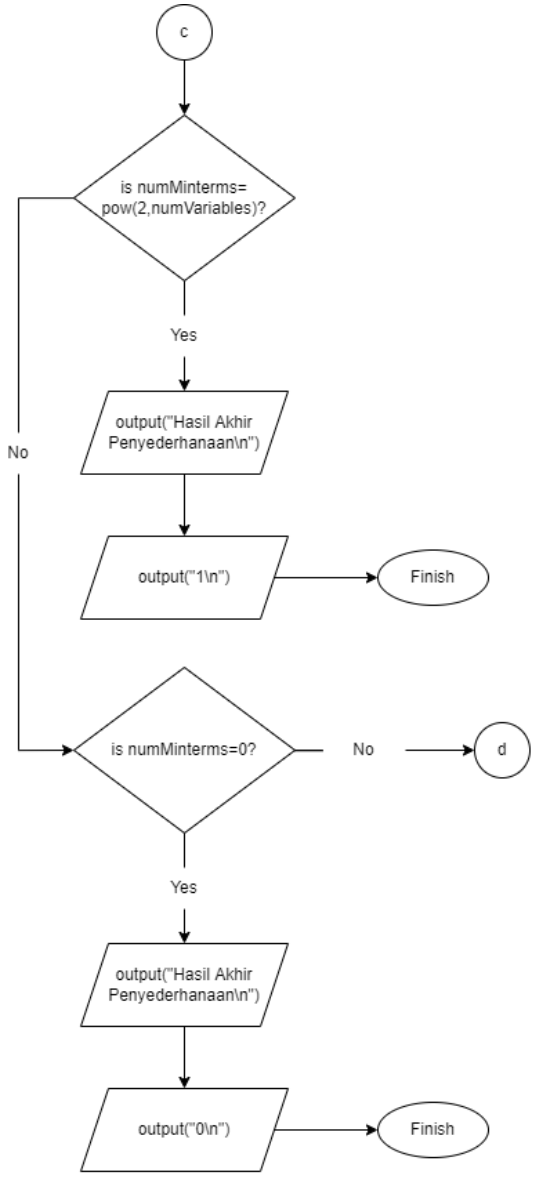
Gambar 12 PrintResult()

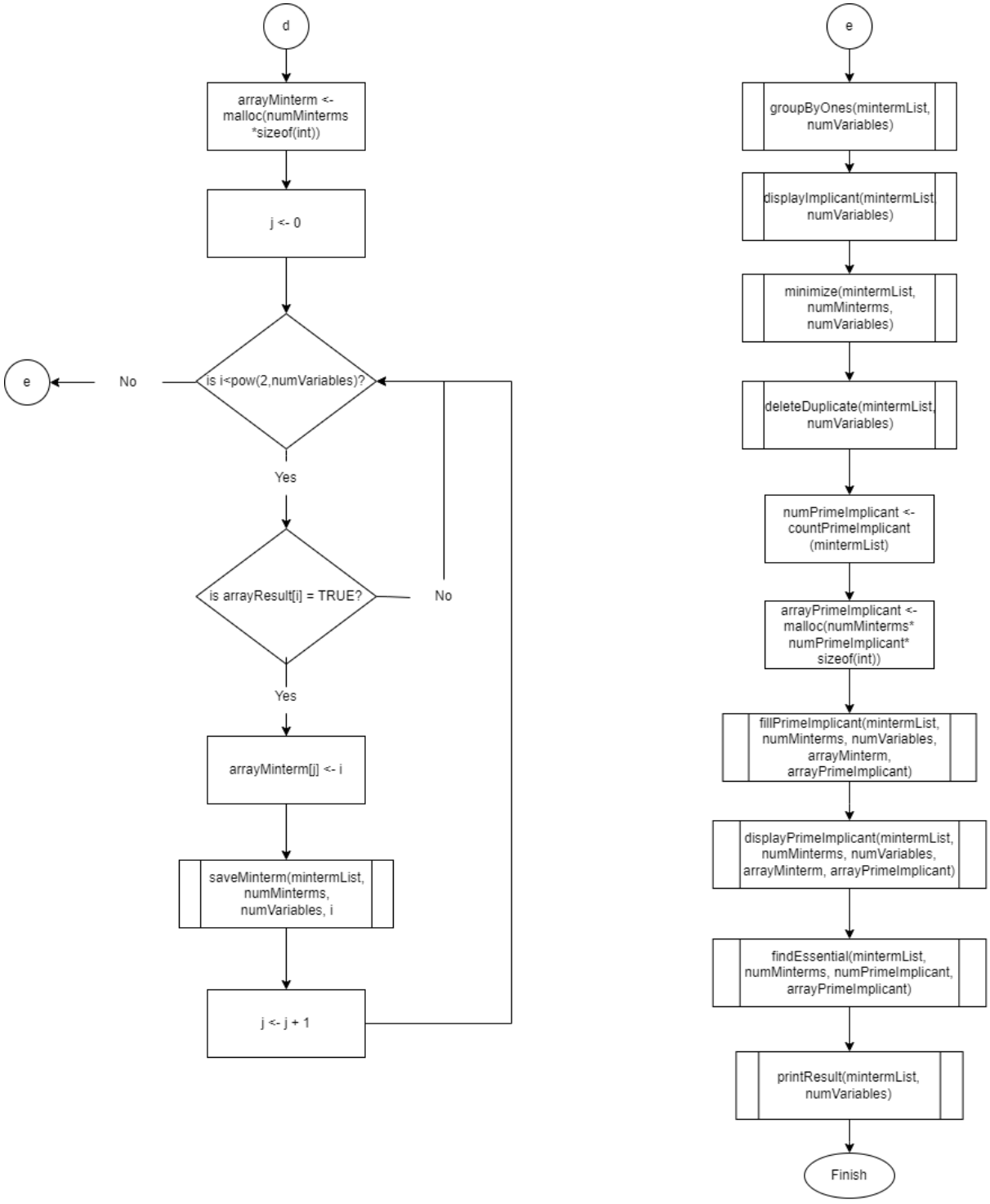
Flowchart Fungsi main()

Fungsi utama untuk menjalankan program. Program akan menampilkan *interface* terlebih dahulu dan menjelaskan fungsi dari program ini. Program akan meminta input data berupa jumlah variabel dan pengisian *truth table* oleh pengguna. Program akan memanfaatkan memori dinamis supaya penggunaan memori lebih efisien. Program akan menampilkan prosedur metode Quine-McCluskey kepada pengguna. Terdapat kasus pengguna memberikan input *truth table* berisi ‘1’. Hal tersebut berarti jumlah *minterm* yang terbentuk akan sama banyaknya dengan *truth table* sehingga program akan langsung menyederhanakan fungsi menjadi ‘1’. Namun apabila pengguna memasukkan *truth table* berisi ‘0’, hal tersebut berarti tidak ada *minterm* untuk kasus tersebut sehingga program akan langsung menampilkan hasil penyederhanaannya adalah ‘0’. Program akan memanggil fungsi saveMinterm untuk menyimpan *minterm* dari *truth table* pengguna. Kemudian *minterm* akan masuk ke dalam *list.* Program akan mengurutkan *minterm* berdasarkan jumlah bit ‘1’ dalam bentuk binernya. Kemudian program akan melakukan penyederhanaan Quine-McCluskey dengan cara memanggil fungsi minimize. Setelah disederhanakan, program akan memanggil deleteDuplicate untuk menyaring kelompok *minterm* yang tergandakan. Program akan mengisi tabel implikan prima berupa *array* dan memanggil fungsi fillPrimeImplicant. Kemudian program akan mencari implikan prima esensial melalui fungsi findEssential dan hasil akan dicetak menggunakan fungsi printResult.


Start
output("Enter the number of variables: ")
input(numVariables)
binary <- malloc(numVariables *sizeof(int))
arrayResult <- malloc(pow(2, numVariables)*sizeof(int)
(LinkedList* mintermList) <- (LinkedList*) malloc(sizeof(LinkedList))
mintermList->head <- NULL
i <- 0
is i<numVariables?
No
a
Yes
output("%c ", (char)91-(numVariables-i))
i <- i + 1
output("| f\n")
a
i <- 0
is i<numVariables*2+3?
No
Yes
output("=")
i <- i + 1
is i<pow(2,numVariables)?
c
No
Yes
is j>=0?
No
b
Yes
binary[j] <- minterm%2
minterm <- minterm/2
j <- j - 1


b
is j<numVariables?
Yes
No
output("%d ", binary[j])
j <- j + 1
output("| ")
input(result)
arrayResult[i] <- result
is result = TRUE?
Yes
numMinterms <-
numMinterms 1


c
is numMinterms=
pow(2,numVariables)?
Yes
No
output("Hasil Akhir
Penyederhanaan\n")
output("1\n")
Finish
is numMinterms=0?
No
d
Yes
output("Hasil Akhir
Penyederhanaan\n")
output("0\n")
Finish

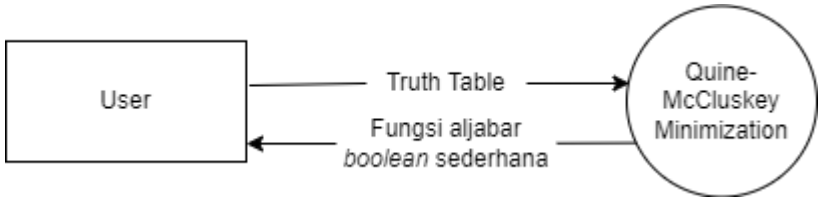


Gambar 13 Main()

DATA FLOW DIAGRAM (DFD)

DFD Level 0

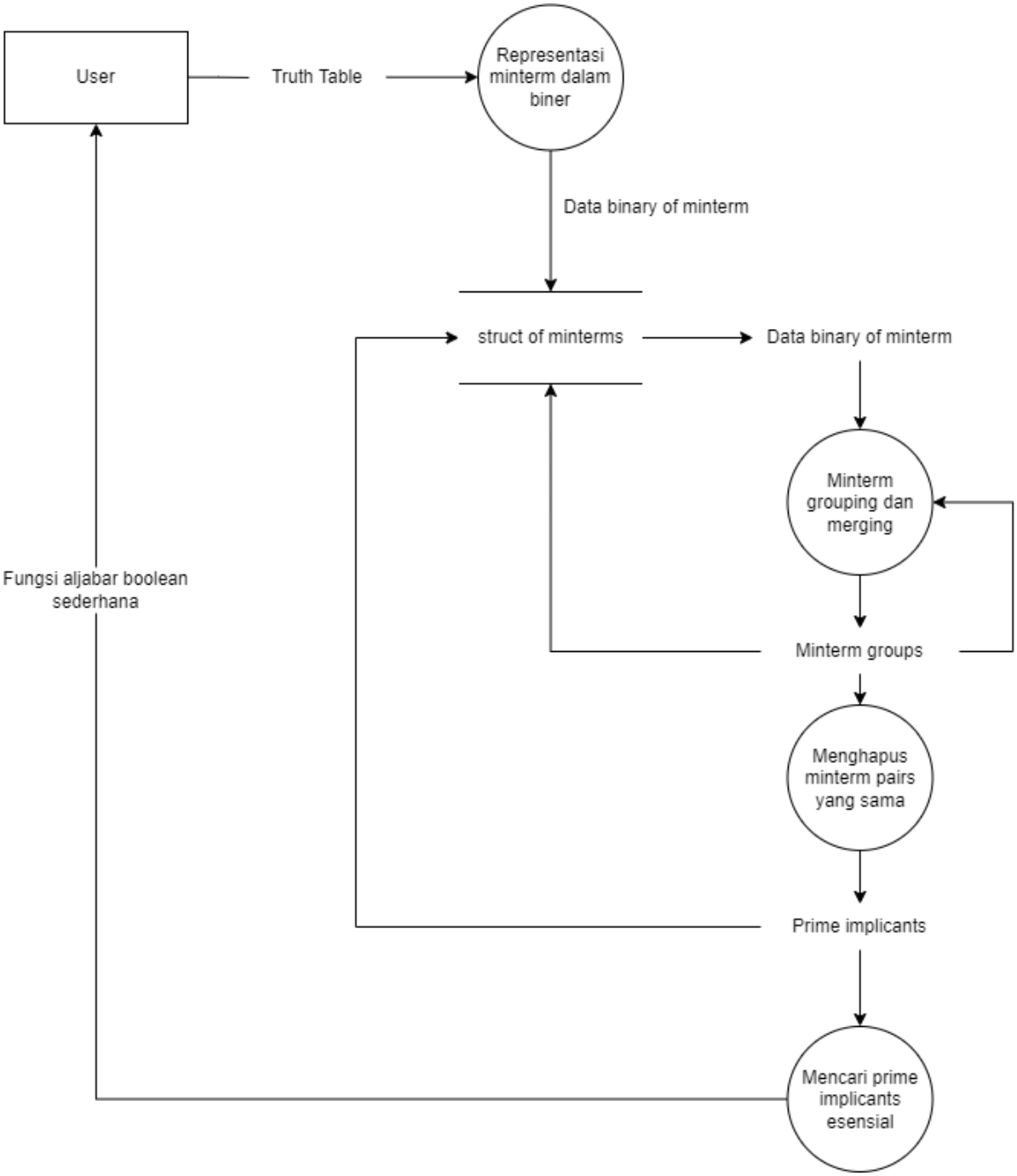
Pengguna akan memberikan input berupa *truth table* kepada program Quine McCluskey *minimization* yang meminta. Data eksternal tersebut akan diproses oleh program Quine-McCluskey kemudian mengembalikan sebuah fungsi aljabar *Boolean* yang telah disederhanakan atau diminimisasi.



Gambar 14 DFD Level 0

DFD Level 1

Pengguna akan memberikan input berupa *truth table* kepada program Quine McCluskey *minimization* yang meminta. Data eksternal tersebut akan diproses oleh program Quine-McCluskey kemudian mengembalikan sebuah fungsi aljabar *Boolean* yang telah disederhanakan atau diminimisasi, seperti pada gambaran DFD level 0. Dalam program Quine-McCluskey, ada beberapa tahap yang akan dilewati data untuk mencapai fungsi aljabar *Boolean* yang sederhana. Data *truth table* akan digunakan untuk membentuk suatu *minterm* yang kemudian akan dikonversi menjadi bentuk biner. *Minterm* akan disusun ke dalam kelompok-kelompok berdasarkan perbandingan salah satu angka binernya yang berbeda. Setelah mendapatkan kelompok *minterm* tersebut. Program akan menyederhanakan kelompok-kelompok tersebut dengan cara menghapus kelompok yang sama sehingga hanya ada kelompok yang berisi *minterm* berbeda-beda.Kemudian program akan membentuk suatu tabel *prime implicant* untuk mencari *prime implicant* yang esensial supaya *prime implicants* yang tersisa merupakan hasil yang paling sederhana. Dengan dicarinya *prime implicant* yang esensial, program akan mengembalikan *prime implicants* tersebut sebagai bentuk paling sederhana fungsi aljabar *Boolean* kepada pengguna.



Gambar 15 DFD Level 1

LOGIC MINIMIZATION DALAM BAHASA PEMROGRAMAN C

Link GitHub *Source Code*

https://github.com/morenzoe/Tugas_Besar_EL2008_Pemecahan_Masalah_dengan_C/blob/main/minimize.c

Source Code

```
/*EL2008 Pemecahan Masalah dengan C 2021/2022
*Tugas              : Tugas Besar 2022
*Kelompok           : 2
*Hari dan Tanggal   : Kamis, 19 Mei 2022
*Nama File          : minimize.c
*Deskripsi          : Program untuk minimisasi logic dengan
*                     Quine-McCluskey Tabular Method. Program
*                     meminta input minterm dalam desimal
*                     dan mengembalikan persamaan hasil penyederhanaan.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct Node{
    int* mintermDec;    // minterm dalam desimal
    int* mintermBin;    // minterm dalam  biner
    int numOnes;        // jumlah bit 1 minterm biner
    int isImplicant;    // indikator minterm implicant, 1 atau 0
    struct Node* next;  // pointer ke node berikutnya
} Node;

typedef struct LinkedList{
    Node* head; // pointer ke node pertama linked list
} LinkedList;
```

```
void red(){
  printf("\033[1;31m");
}

void reset(){
  printf("\033[0m");
}

void printLinkedList(LinkedList* list, int numVariables) {
    int i;

    Node* temp = list->head;

    printf("\n");
    while (temp != NULL) {
        i = 1;
        printf("mintermDec: %d", temp->mintermDec[0]);
        while((temp->mintermDec[i]!=-1)&&(i<(numVariables*numVariables))){
            printf(", %d", temp->mintermDec[i]);
            i += 1;
        }
        printf("\n");
        printf("mintermBin: ");
        for(i=0; i<numVariables; ++i){
            if(temp->mintermBin[i]==-2){
                printf("-");
            } else{
                printf("%d", temp->mintermBin[i]);
            }
        }
        printf("\n");
        printf("numOnes: %d\n", temp->numOnes);
        printf("isImplicant: %d\n", temp->isImplicant);
        printf("\n");
```

```
        temp = temp->next;
    }
    printf("\n");

    free(temp);

    return;
}

int countOnes(Node* node, int numVariables){
    int i;
    int sum = 0;    // variabel jumlah bit 1

    // Memeriksa semua bit
    for(i=0; i<numVariables; ++i){
        // Menambahkan jumlah bit 1
        if(node->mintermBin[i]==1){
            sum += 1;
        }
    }

    return sum;
}

void insertNode(LinkedList* list, Node* node){
    // Jika linked list kosong, assign node sebagai head
    if(list->head==NULL){
        list->head = node;
    } else{
        // Membuat pointer sementara untuk menyusuri linked list
        struct Node* temp = list->head;

        // Menyusuri linked list hingga node terakhir
        while(temp->next!=NULL){
```

```
            temp = temp->next;
        }

        // Menghubungkan node terakhir dengan node baru
        temp->next = node;
    }

    return;
}

void saveMinterm(LinkedList* list, int numMinterms, int numVariables, int
minterm){
    int i;

    // Mengalokasikan memori node baru
    Node* new = (Node*) malloc(sizeof(Node));

    // Mengisi data node baru
    // data minterm dalam desimal
    new->mintermDec = malloc(numMinterms*sizeof(int));
    new->mintermDec[0] = minterm;
    // Menginisialisasi sisa array minterm dengan -1
    for(i=1; i<numMinterms; ++i){
        new->mintermDec[i] = -1;
    }

    // data minterm dalam biner
    new->mintermBin = malloc(numVariables*sizeof(int));
    // Konversi dan simpan minterm dalam bentuk biner
    for(i=numVariables-1; i>=0; --i){
        new->mintermBin[i] = minterm%2;
        minterm = minterm/2;
    }

    // data jumlah bit 1
```

```
    new->numOnes = countOnes(new, numVariables);

    // data indikator implicant, asumsi setiap minterm adalah implicant
    new->isImplicant = 1;

    // data node berikutnya
    new->next = NULL;

    // Menambahkan node baru ke linked list minterm
    insertNode(list, new);

    return;
}

void groupByOnes(LinkedList* list, int numVariables){
    int i;
    int temp;   // variabel sementara untuk pertukaran

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp1 = list->head;
    Node* temp2;

    // Menyusuri linked list
    while(temp1!=NULL){
        // Membandingkan temp1 dengan semua node setelahnya
        temp2 = temp1->next;
        while(temp2!=NULL){
            // Menukar data kedua node
            if((temp1->numOnes)>(temp2->numOnes)){
                // data minterm dalam desimal
                temp = temp1->mintermDec[0];
                temp1->mintermDec[0] = temp2->mintermDec[0];
                temp2->mintermDec[0] = temp;
```

```
                // data minterm dalam biner
                for(i=0; i<numVariables; ++i){
                    temp = temp1->mintermBin[i];
                    temp1->mintermBin[i] = temp2->mintermBin[i];
                    temp2->mintermBin[i] = temp;
                }

                // data jumlah bit 1
                temp = temp1->numOnes;
                temp1->numOnes = temp2->numOnes;
                temp2->numOnes = temp;
            }

            // Melanjutkan perbandingan ke node berikutnya
            temp2 = temp2->next;
        }

        // Melanjutkan perbandingan ke node berikutnya
        temp1 = temp1->next;
    }

    return;
}

void displayImplicant(LinkedList* list, int numVariables){
    int i;
    int numGroup = list->head->numOnes;   // variabel nomor baris,
inisialisasi jumlah bit 1 implicant pertama

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Mencetak judul tabel
    printf("Group No.\tMinterms\tBinary of Minterms\n");
    printf("================================================\n");
```

```
// Mencetak nomor grup
printf("     %d:\n", numGroup);

// Menyusuri linked list
while(temp!=NULL){
    // Jika mencapai implicant dengan jumlah bit 1 berbeda
    if(numGroup!=temp->numOnes){
        // Mengubah nomor grup
        numGroup = temp->numOnes;

        // Mencetak pembatas grup
        printf("-------------------------------------------------\n");

        // Mencetak nomor grup
        printf("     %d:\n", numGroup);
    }

    // Mencetak isi grup
    // mencetak minterm dalam desimal
    printf("\t\t%d", temp->mintermDec[0]);
    i = 1;
    while((temp->mintermDec[i]!=-1)&&(i<(numVariables*numVariables))){
        printf(",%d", temp->mintermDec[i]);
        i += 1;
    }
    printf("\t\t");

    // mencetak minterm dalam biner
    for(i=0; i<numVariables; ++i){
        if(temp->mintermBin[i]==-2){
            printf("-");
        } else{
            printf("%d", temp->mintermBin[i]);
        }
```

```
        }
        printf("\n");

        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
    }
    printf("================================================\n\n");

    return;
}

void minimize(LinkedList* list, int numMinterms, int numVariables){
    int i;
    int j;
    int simplified;     // variabel penanda adanya implicant yang
disederhanakan
    int notSimplified;  // variabel penanda adanya implicant yang tidak
disederhanakan
    int idxChange;      // variabel indeks bit yang berubah
    int sumChange;      // variabel jumlah perubahan

    int step=1;         // debugging

    while(simplified){
        // Menginisialisasi penanda
        simplified = 0;
        notSimplified = 0;

        // Membuat linked list untuk menyimpan minterm hasil penyederhanaan
        LinkedList* mintermGroupList = (LinkedList*)
malloc(sizeof(LinkedList));
        mintermGroupList->head = NULL;

        // Membuat pointer sementara untuk menyusuri linked list
        Node* temp1 = list->head;
        Node* temp2;
```

```
// Menyusuri linked list
while(temp1!=NULL){
    // debugging
    //printf("masuk while temp1\n");

    // Membandingkan temp1 dengan semua node setelahnya
    temp2 = temp1->next;
    while(temp2!=NULL){
        // debugging
        //printf("masuk while temp2\n");

        // Menginisialisasikan jumlah perubahan
        sumChange = 0;

        // Melakukan perbandingan dua grup yang bersebelahan
        if((temp2->numOnes)-(temp1->numOnes)==1){
            // Melakukan perbandingan antar tiap bit
            for(i=0; i<numVariables; ++i){
                // Jika 1 bit berbeda
                if((temp1->mintermBin[i])!=(temp2->mintermBin[i])){
                    // Menambah jumlah perubahan bit
                    sumChange += 1;

                    // Menyimpan indeks perubahan bit
                    idxChange = i;
                }
            }
        }

        // debugging
        //printf("sumChange: %d\n", sumChange);

        // Jika hanya terdapat 1 perubahan bit
```

```
if(sumChange==1){
    // debugging
    //printf("hanya 1 perubahan bit\n");

    // Menandai ada penyederhanaan
    simplified = 1;

    // Mengubah indikator minterm bukan implicant
    temp1->isImplicant = 0;
    temp2->isImplicant = 0;

    // Menambahkan node penyederhanaan ke linked list
    Node* new = (Node*) malloc(sizeof(Node));

    // Mengisi data node baru
    // data minterm dalam desimal
    new->mintermDec = malloc(numMinterms*sizeof(int));

    // dari node pertama
    i=0;
    while(temp1->mintermDec[i]!=-1){
        new->mintermDec[i] = temp1->mintermDec[i];
        i += 1;
    }

    // dari node kedua
    j=0;
    while(temp2->mintermDec[j]!=-1){
        new->mintermDec[i] = temp2->mintermDec[j];
        i += 1;
        j += 1;
    }

    // Mengisi sisa array minterm dengan -1
```

```
                for(j=i; j<numMinterms; ++j){
                    new->mintermDec[j] = -1;
                }

                // data minterm dalam biner
                new->mintermBin = malloc(numVariables*sizeof(int));
                for(i=0; i<numVariables; ++i){
                    new->mintermBin[i] = temp1->mintermBin[i];
                }
                // Menandai bit yang berubah
                new->mintermBin[idxChange] = -2;

                // data jumlah bit 1
                new->numOnes = temp1->numOnes;

                // data node berikutnya
                new->next = NULL;

                // Menambahkan node baru ke linked list penyederhanaan
                insertNode(mintermGroupList, new);
            }

            // Melanjutkan perbandingan ke node berikutnya
            temp2 = temp2->next;
        }

        // Melanjutkan perbandingan ke node berikutnya
        temp1 = temp1->next;
    }

    // Menambahkan minterm yang tidak disederhanakan
    temp1 = list->head;
    while(temp1!=NULL){
        // Jika minterm implicant
```

```
            if(temp1->isImplicant==1){
                notSimplified = 1;
                // Mengalokasikan memori node baru
                Node* new = (Node*) malloc(sizeof(Node));

                // Mengisi data node baru
                // data minterm dalam desimal
                new->mintermDec = malloc(numMinterms*sizeof(int));
                new->mintermDec[0] = temp1->mintermDec[0]; ;
                // Menginisialisasi sisa array minterm dengan -1
                for(i=1; i<numMinterms; ++i){
                    new->mintermDec[i] = -1;
                }

                // data minterm dalam biner
                new->mintermBin = malloc(numVariables*sizeof(int));
                // Konversi dan simpan minterm dalam bentuk biner
                for(i=numVariables-1; i>=0; --i){
                    new->mintermBin[i] = temp1->mintermBin[i];
                }

                // data jumlah bit 1
                new->numOnes = temp1->numOnes;

                // data indikator implicant, asumsi setiap minterm adalah
implicant
                new->isImplicant = 1;

                // data node berikutnya
                new->next = NULL;

                // Menambahkan node baru ke linked list minterm
                insertNode(mintermGroupList, new);
            }
```

```
            // Melanjutkan pengisian node berikutnya
            temp1 = temp1->next;
        }

        // Jika mintermGroupList terisi
        if(simplified||notSimplified){
            // Mengubah linked list minterm menjadi linked list hasil
penyederhanaan
            list->head = mintermGroupList->head;

            // Menampilkan tabel hasil penyederhanaan
            red();
            printf("Tabel Penyederhanaan ke-%d\n", step);
            reset();
            displayImplicant(list, numVariables);

            step += 1;
        }
    }

    return;
}

void deleteDuplicate(LinkedList* list, int numVariables){
    int i;
    int same;   // variabel penanda dua prime implicant sama

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Menyusuri linked list
    while((temp!=NULL)&&(temp->next!=NULL)){
        // Menginisialisasi penanda
        same = 1;
```

```
            // Melakukan perbandingan antar tiap bit
            for(i=0; i<numVariables; ++i){
                // Jika terdapat bit yang berbeda
                if((temp->mintermBin[i])!=(temp->next->mintermBin[i])){
                    // Mengubah penanda
                    same = 0;
                }
            }

            // Jika kedua prime implicant sama
            if(same){
                // Lewati prime implicant yang sama
                temp->next = temp->next->next;
            }

            // Melanjutkan perbandingan ke node berikutnya
            temp = temp->next;
        }

        // Menampilkan tabel hasil penyederhanaan
        red();
        printf("Tabel Hasil Penghapusan Duplikat\n");
        reset();
        displayImplicant(list, numVariables);

        return;
    }

    int countPrimeImplicant(LinkedList* list){
        int count = 0;  // variabel penyimpan jumlah prime implicant

        // Membuat pointer sementara untuk menyusuri linked list
        Node* temp = list->head;
```

```
    // Menyusuri linked list
    while(temp!=NULL){
        // Menambah jumlah prime implicant
        count += 1;

        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
    }

    return count;
}

void fillPrimeImplicant(LinkedList* list, int numMinterms, int
numVariables, int* arrayMinterm, int* arrayPrimeImplicant){
    int i=0;
    int j;
    int k;

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Menyusuri linked list
    while(temp!=NULL){
        // Mengisi matriks prime implicant
        j = 0;
        while((temp->mintermDec[j]!=-1)&&(j<(numVariables*numVariables))){
            k = 0;
            while(temp->mintermDec[j]!=arrayMinterm[k]){
                k += 1;
            }
            arrayPrimeImplicant[i*numMinterms+k] = 1;

            j += 1;
        }
        i += 1;
```

```
        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
    }

    return;
}

void displayPrimeImplicant(LinkedList* list, int numMinterms, int
numVariables, int* arrayMinterm, int* arrayPrimeImplicant){
    int i;
    int j;
    int k=0;

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Mencetak minterm dalam desimal
    printf("  ");
    for(i=0; i<numMinterms; ++i){
        printf("%d\t", arrayMinterm[i]);
    }
    printf("|");

    // Mencetak judul tabel
    printf("\tMinterms\n");

    for(i=0; i<numMinterms*11; ++i){
        printf("=");
    }
    printf("\n");

    // Menyusuri linked list
    while(temp!=NULL){
        // Mencetak minterm di dalam prime implicant
```

```
        printf("  ");
        for(i=0; i<numMinterms; ++i){
            if(arrayPrimeImplicant[k*numMinterms+i]==1){
                printf("X\t");
            } else{
                printf(" \t");
            }
        }
        printf("|\t");

        // Mencetak prime implicant
        printf("%d", temp->mintermDec[0]);
        i = 1;
        while((temp->mintermDec[i]!=-1)&&(i<(numVariables*numVariables))){
            printf(",%d", temp->mintermDec[i]);
            i += 1;
        }

        printf("\n");

        k += 1;

        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
    }
    for(i=0; i<numMinterms*11; ++i){
        printf("=");
    }
    printf("\n");

    return;
}

void findEssential(LinkedList* list, int numMinterms, int
numPrimeImplicant, int* arrayPrimeImplicant){
```

```
    int i=0;
    int j;
    int k;
    int count;
    int isEssential;

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Menyusuri linked list
    while(temp!=NULL){
        isEssential = 0;
        // cek semua baris
        for(j=0; j<numMinterms; ++j){
            count = 0;
            // jika ada isi 1, cek kolom
            if(arrayPrimeImplicant[i*numMinterms+j]==1){
                for(k=0; k<numPrimeImplicant; ++k){
                    if(arrayPrimeImplicant[k*numMinterms+j]==1){
                        count += 1;
                    }
                }

                // kalau dalam satu kolom hanya ada 1 bit 1, prime
implicant adalah essential
                if(count==1){
                        temp->isImplicant = 1;
                    }
            }
        }

        i += 1;

        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
```

```
    }

    return;
}

void printResult(LinkedList* list, int numVariables){
    int i;

    // Membuat pointer sementara untuk menyusuri linked list
    Node* temp = list->head;

    // Menyusuri linked list
    while(temp!=NULL){
        // Jika prime implicant adalah essential
        if(temp->isImplicant==1){
            for(i=0; i<numVariables; ++i){
                if(temp->mintermBin[i]==1){
                    printf("%c", (char)91-(numVariables-i));
                } else if(temp->mintermBin[i]==0){
                    printf("%c'", (char)91-(numVariables-i));
                }
            }

            if(temp->next!=NULL){
                printf(" + ");
            }
        }

        // Melanjutkan perbandingan ke node berikutnya
        temp = temp->next;
    }

    printf("\n");
```

```
    return;
}

int main()
{
    int i;
    int j;
    int numVariables;           // variabel penyimpan jumlah variabel
    int numMinterms=0;          // variabel penyimpan jumlah minterm

    int* binary;                // array hasil konversi desimal ke biner
    int result;                 // variabel penyimpan input hasil truth
table
    int* arrayResult;           // array penyimpan semua input hasil truth
table

    int minterm;                // variabel penyimpan minterm input
    int* arrayMinterm;          // array penyimpan semua minterm input
    int numPrimeImplicant;      // variabel penyimpan jumlah prime
implicant
    int* arrayPrimeImplicant;   // matriks penyimpan semua prime implicant
dan mintermnya

    // Mencetak informasi awal
    red();
    printf("Selamat datang di Algebra Boolean Calculator!\n\n");
    reset();
    printf("Program ini dapat menyederhanakan\n");
    printf("persamaan boolean dari input truth table\n\n");

    // Meminta input data
    printf("Jumlah variabel adalah integer yang\n");
    printf("lebih besar dari 0 dan kurang dari 27\n");
    printf("Masukkan jumlah variabel: ");
    scanf("%d",&numVariables);
```

```
// Mengalokasikan memori array biner
binary = malloc(numVariables*sizeof(int));

// Mengalokasikan memori array result
arrayResult = malloc(pow(2,numVariables)*sizeof(int));

// Membuat linked list untuk menyimpan minterm
LinkedList* mintermList = (LinkedList*) malloc(sizeof(LinkedList));
mintermList->head = NULL;

// Menampilkan judul tabel
// menampilkan variabel
red();
printf("\nTruth Table\n");
reset();
for(i=0; i<numVariables; ++i){
    printf("%c ", (char)91-(numVariables-i));
}
printf("| f");
printf("\n");

// menampilkan pembatas
for(i=0; i<numVariables*2+3; ++i){
    printf("=");
}
printf("\n");

for(i=0; i<pow(2,numVariables); ++i){
    int minterm = i;
    for(j=numVariables-1; j>=0; --j){
        binary[j] = minterm%2;
        minterm = minterm/2;
    }
```

```
    for(j=0; j<numVariables; ++j){
        printf("%d ", binary[j]);
    }
    printf("| ");

    scanf("%d", &result);
    arrayResult[i] = result;

    if(result){
        numMinterms += 1;
    }
}
printf("\n");

// Jika minterm sebanyak baris truth table
if(numMinterms==pow(2,numVariables)){
    red();
    printf("Hasil Akhir Penyederhanaan\n");
    reset();
    printf("1\n");

    return 0;

// Jika tidak ada minterm
} else if(numMinterms==0){
    red();
    printf("Hasil Akhir Penyederhanaan\n");
    reset();
    printf("0\n");

    return 0;
}

// Meminta input minterm dalam desimal
```

```
// asumsi input terurut dari minterm terkecil
arrayMinterm =  malloc(numMinterms*sizeof(int));
j =0;
for(i=0; i<pow(2,numVariables); ++i){
    if(arrayResult[i]){

        // Menyimpan minterm ke array
        arrayMinterm[j] = i;

        // Menyimpan minterm ke mintermList
        saveMinterm(mintermList, numMinterms, numVariables, i);

        j += 1;
    }
}

// Mengurutkan mintermList berdasarkan jumlah bit 1 minterm dalam biner
groupByOnes(mintermList, numVariables);

// Menampilkan tabel minterm sebelum penyederhanaan
red();
printf("Tabel Hasil Pengelompokkan\n");
reset();
displayImplicant(mintermList, numVariables);

// Melakukan penyederhanaan dengan Quine-McCluskey Tabular Method
minimize(mintermList, numMinterms, numVariables);

// Menghapus prime implicant duplikat
deleteDuplicate(mintermList, numVariables);

// Menghitung jumlah prime implicant
numPrimeImplicant = countPrimeImplicant(mintermList);
```

```
    // Mengisi tabel prime implicant
    arrayPrimeImplicant =
malloc(numMinterms*numPrimeImplicant*sizeof(int));
    fillPrimeImplicant(mintermList, numMinterms, numVariables,
arrayMinterm, arrayPrimeImplicant);


    // Menampilkan tabel prime implicant
    red();
    printf("Tabel Implikan Prima\n");
    reset();
    displayPrimeImplicant(mintermList, numMinterms, numVariables,
arrayMinterm, arrayPrimeImplicant);


    // Mencari essential prime implicant
    findEssential(mintermList, numMinterms, numPrimeImplicant,
arrayPrimeImplicant);


    // Mencetak hasil akhir penyederhanaan
    red();
    printf("\nHasil Akhir Penyederhanaan\n");
    reset();
    printResult(mintermList, numVariables);


    return 0;


}
```

KESIMPULAN DAN *LESSON LEARNED*

Program penyederhanaan logika fungsi aljabar *Boolean* dalam bahasa pemrograman C dapat dijalankan dan mendapat hasil output yang sesuai. Metode Quine-McCluskey dapat diimplementasikan dalam kode berbahasa C. *Logic minimization* tersebut memanfaatkan metode tabulasi menggunakan suku *minterms* untuk mendapatkan jumlahnya dan pengelompokan sampai membentuk implikan prima esensial yang menunjukkan fungsi aljabar *Boolean* tersebut telah disederhanakan. Aplikasi metode Quine-McCluskey pada program bahasa C memanfaatkan *struct* dan beberapa *looping* untuk menyederhanakan fungsi aljabar *Boolean*. Algoritma *logic minimization* sangat mementingkan kompleksitas waktu dan ruang terutama jika variable fungsi tersebut besar. Dengan adanya pemakaian *struct*, algoritma yang digunakan menjadi lebih efisien.

Proses penyederhanaan ini dapat disimpulkan berhasil dilakukan karena fungsi aljabar *Boolean* yang rumit dapat disederhanakan menjadi fungsi paling sederhana mengikuti aturan *Sum of Product*. Secara manual, penyelesaian fungsi pada masukan membutuhkan waktu lebih lama karena banyaknya implikan dalam fungsi tersebut. Namun, dengan penyederhanaan fungsi, implikan menjadi lebih sedikit sehingga perhitungan secara manual dapat dilakukan lebih cepat. Hasil yang didapatkan juga menunjukkan hasil yang sama sehingga penggunaan metode Quine-McCluskey sangat bermanfaat dan cocok. Dengan adanya minimisasi, pengguna dapat menghemat waktu dan juga biaya pembuatan sirkuit logika dengan baik. Program minimisasi logika telah berhasil dijalankan.

PEMBAGIAN TUGAS

Pembagian Tugas *Source Code*

Eraraya Morenzo Muten/18320003	Menyusun program.

Pembagian Tugas Dokumen Laporan

Michelle Angelina/18320007	Menulis laporan akhir.

Pembagian Tugas Bahan Presentasi

Shadrina Syahla Vidyana/18320031	Membuat bahan presentasi akhir.

REFERENSI

1. https://sourceforge.net/projects/mini-qmc/, diakses 1 Mei 2022, 19:03 WIB.
2. https://www.tutorialspoint.com/digital_circuits/digital_circuits_quine_mccluskey_tabular_method.htm, diakses 1 Mei 2022, 19:27 WIB.
3. https://www.geeksforgeeks.org/quine-mccluskey-method/, diakses 1 Mei 2022, 20:38 WIB.
4. https://ecomputernotes.com/what-is-c/operator/boolean-operators, diakses 2 Mei 2022, 13:43 WIB.
5. https://www.javatpoint.com/c-program-to-convert-decimal-to-binary, diakses 5 Mei 2022, 08:12 WIB.
6. https://www.tutorialspoint.com/cprogramming/c_return_arrays_from_function.htm, diakses 13 Mei 2022, 10:30 WIB.
7. http://generalnote.com/C-Programm/Programs-on-Array/C-Program-to-insert-&-Display-the-element-in-2D-Array.php, diakses 15 Mei 2022, 13:02 WIB.
8. https://www.geeksforgeeks.org/minimization-of-boolean-functions/, diakses 19 Mei 2022, 04:43 WIB.
9. https://www.sciencedirect.com/topics/computer-science/logic-minimization, diakses 20 Mei 2022, 23:29 WIB.