

TUGAS BESAR EL2008 PEMECAHAN MASALAH DENGAN C
ALGORITMA UNTUK MEMINIMISASI LOGIC DENGAN MENGGUNAKAN
METODE TABULAR (Quine-McCluskey)

*Disajikan untuk memenuhi tugas besar Mata Kuliah EL2008 Pemecahan Masalah dengan C
yang diampu oleh:*

Muhammad Ogin Hasanuddin, S.T., M.T.
Dr. Arif Sasongko, S.T., M.T.



Diusulkan oleh:

Farhan Hakim Iskandar	13220007
Fitra Nurindra	13220011
Muhammad Daffa Daniswara	13220043

INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2022

DAFTAR ISI

	Halaman
DAFTAR ISI	ii
1. LATAR BELAKANG PERMASALAHAN DAN EKSPLORASI ALGORITMA MINIMISASI LOGIC.....	1
1.1 Latar Belakang Permasalahan	1
1.2 Eksplorasi Algoritma Minimisasi Logic	1
2. RANCANGAN PROGRAM SIMULASI.....	3
2.1 Deskripsi Simulasi	3
2.2 Flowchart	5
2.2 Data Flow Diagram (DFD)	5
3. IMPLEMENTASI PROGRAM DALAM BAHASA C	6
3.1 Link GitHub Source Code	6
3.2 Simulasi Program	6
3.3 Source Code	16
3.3.1 Source Code minimisasi_lib.h	16
3.3.2 Source Code minimisasi_lib.c	19
3.3.3 Source Code main.c	23
4. KESIMPULAN DAN <i>LESSON LEARNED</i>	31
5. PEMBAGIAN TUGAS DALAM KELOMPOK	32
DAFTAR REFERENSI.....	33

1. LATAR BELAKANG PERMASALAHAN DAN EKSPLORASI ALGORITMA MINIMISASI LOGIC

1.1 Latar Belakang Permasalahan

Secara umum, permasalahan yang terdapat pada tugas besar Mata Kuliah Pemecahan Masalah Dengan C (EL2008) adalah mengenai minimisasi fungsi logika. Sehingga secara garis besar, kita diminta untuk membuat program yang dapat melakukan minimisasi suatu fungsi logika. Minimisasi fungsi logika merupakan suatu proses menyederhanakan suatu fungsi *boolean algebra*. Setiap fungsi *boolean algebra* dapat dinyatakan dalam bentuk *sum of product* (minterm) atau *product of sum* (maxterms). Dalam melakukan penyederhanaan fungsi ekspresi *boolean algebra* dapat menggunakan beberapa cara, diantaranya yaitu metode manipulasi *boolean algebra*, Karnaugh Maps, dan Quine-McCluskey atau Metode Tabular. Beberapa literatur merekomendasikan penggunaan metode Quine-McCluskey karena dianggap paling baik dalam penyederhanaan dan dapat digunakan untuk variabel fungsi *boolean algebra* yang besar. Metode tabulasi menggunakan tahap-tahap penyederhanaan yang jelas dan teratur dalam penyederhanaan suatu fungsi aljabar.

Minimisasi logika fungsi aljabar Boolean diperlukan untuk mengurangi penggunaan kompleksitas sirkuit supaya menghasilkan program atau sirkuit yang efisien untuk digunakan. Selain itu, penyederhanaan fungsi *boolean algebra* juga dapat mengurangi biaya dan penggunaan *logic gate* pada suatu sirkuit atau rangkaian tanpa mengubah hasil keluaran rangkaian tersebut. Penyederhanaan fungsi logika 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 tidak dioptimalisasi, barang-barang digital yang digunakan saat ini tidak akan secepat itu dan lebih mahal karena komponen-komponennya yang tidak disederhanakan.

1.2 Eksplorasi Algoritma Minimisasi Logic

Logic minimization adalah proses menyederhanakan ekspresi boolean. Tujuan dari *logic minimization* ialah untuk memudahkan serta meningkatkan efisiensi suatu sistem. Pada saat eksplorasi metode, kami mencoba beberapa algoritma *logic minimization*, yakni *Boolean Algebra*, Karnaugh Map, dan Quine-McCluskey atau Metode Tabular. Kelompok kami memutuskan untuk menggunakan algoritma Quine-McCluskey untuk diimplementasikan ke dalam program yang akan kelompok kami buat untuk menyelesaikan permasalahan yang terdapat dalam tugas besar kali ini.

Algoritma Quine-McCluskey ini dipilih karena jika dibandingkan dengan metode lain, metode ini lebih mudah diimplementasikan ke dalam program dan direalisasikan secara komputasional. Hal ini disebabkan karena metode ini dilakukan banyak pengulangan atau *looping*. Selain itu, algoritma ini juga cocok untuk penyederhanaan logic dengan variabel yang banyak. Tahapan metode tabular ini adalah penentuan *prime implicant* dengan mencari *matched pairs* dari minterm. Selanjutnya penentuan *minimum cover*, yakni memilih *prime implicant* yang telah didapat untuk mencakup semua suku dan menghasilkan fungsi yang paling sederhana (*essential prime implicant*).

Secara umum, algoritma Quine-McCluskey didasarkan atas prinsip reduksi, misalkan terdapat suatu fungsi *boolean algebra* sebagai berikut.

$$f(A, B) = AB + AB' = A$$

Dalam fungsi tersebut, A dapat berupa variabel atau tim dan B adalah variabel. Hal tersebut berarti bahwa ketika dua buah minterm mengandung variabel yang sama hanya berbeda dalam satu variabel, maka mereka bisa digabungkan bersama dan membentuk suatu ekspresi baru yang lebih sederhana. Kemudian proses reduksi tersebut akan terus dilakukan sampai tidak ada suku yang dapat digabungkan lagi. Istilah lain yang menandakan bahwa beberapa minterm tidak dapat direduksi lagi yaitu *prime implicant* (PI). Langkah terakhir dari algoritma QM ini adalah memilih set yang terdapat di dalam PI yang mengandung paling sedikit kemungkinan jumlah PI dan mencakup semua minterm yang terdapat dalam ekspresi fungsi *boolean algebra*. PI yang dipilih disebut dinamakan *essential prime implicant* (EPI). EPI tersebut mewakili ekspresi akhir dari fungsi *boolean algebra* yang paling minimum.

2. RANCANGAN PROGRAM SIMULASI

2.1 Deskripsi Simulasi

Oleh karena algoritma yang kelompok kami gunakan adalah algoritma Quine-McCluskey, maka dapat diketahui bahwa *input* yang akan diberikan dari program simulasi yang akan kami buat adalah berupa informasi mengenai ekspresi fungsi *boolean algebra* yang ingin dicari ekspresi minimumnya.

Misalkan diberikan suatu fungsi *boolean algebra* sebagai berikut.

$$f(A, B, C, D) = \sum m(1, 3, 7, 12, 13, 14, 15)$$

Dari ekspresi fungsi *boolean algebra* tersebut, kelompok kami memutuskan bahwa *input* yang perlu diberikan oleh *user* untuk memberikan informasi mengenai ekspresi fungsi *boolean algebra* yang ingin dicari ekspresi minimumnya yaitu:

- Banyaknya variabel dari fungsi *boolean algebra*.
- Banyaknya minterm secara keseluruhan (termasuk *don't care* minterm).
- Banyaknya *don't care* minterm.
- Indeks minterm secara keseluruhan (termasuk *don't care* minterm) dalam bentuk desimal.
- Indeks *don't care* minterm dalam bentuk desimal.

Dalam setiap pemberian *input* program simulasi yang dibuat diharapkan dapat memvalidasi nilai *input* yang diberikan agar program simulasi dapat berjalan dengan baik. Validasi tersebut diantaranya:

- Banyaknya variabel maksimal dari fungsi *boolean algebra* hanya sebanyak 10 variabel dan jumlah variabel minimalnya adalah sebanyak 1 variabel.
- Banyaknya minterm keseluruhan haruslah ($0 < Tminterm \leq 2^{nVariable}$).
- Banyaknya *don't care* minterm haruslah ($0 \leq DCminterm < Tminterm$).
- Indeks minterm yang diberikan harus dimulai dari indeks terkecil hingga indeks terbesar dan indeks minterm haruslah ($0 \leq IdxMT < 2^{nVariable}$).
- Indeks *don't care* minterm yang diberikan harus dimulai dari indeks terkecil hingga indeks terbesar dan indeks *don't care* minterm haruslah ($0 \leq IdxDCMT < 2^{nVariable}$).

Kemudian setelah *input* yang diberikan sudah valid, maka program simulasi akan memulai proses minimisasi fungsi logika. Proses awal yang akan dilakukan yaitu mengubah indeks minterm yang telah diberikan dalam bentuk desimal menjadi ke dalam bentuk binary. Kemudian algoritma Quine-McCluskey pun akan mulai dilakukan. Secara umum proses yang terdapat dalam program simulasi yang akan dibuat oleh kelompok kami adalah sebagai berikut (penjelasan algoritma lebih lanjut akan terdapat pada bagian *flowchart* dan *data flow diagram*).

1. Langkah awal akan dibuat sebuah tabel (tabel 0) yang berisi minterm yang telah dikelompokkan ke dalam beberapa grup berdasarkan banyaknya angka 1 dalam representasi binary-nya. Untuk contoh ekspresi fungsi *boolean algebra* di atas akan di dapatkan tabel sebagai berikut.

Group	Minterm	Representasi Binary (ABCD)
Group 0	m1	0001
Group 1	m3	0011
	m12	1100
Group 2	m7	0111
	m13	1101

	m14	1110
Group 3	m15	1111

Tabel 2.1. Tabel Inisialisasi

2. Kemudian antara 2 buah minterm yang memiliki 1 buah perbedaan akan saling dipasangkan dan hasilnya akan disimpan dalam sebuah tabel (tabel 1). Tabel yang diperoleh adalah sebagai berikut.

Group	Matched-Pairs	Representasi Binary (ABCD)
Group 0	m1-m3	00X1
Group 1	m3-m7	0X11
	m12-m13	110X
	m12-m14	11X0
Group 2	m7-m15	X111
	m13-m15	11X1
	m14-m15	111X

Tabel 2.2. Tabel Reduksi 1

3. Kemudian akan dicari kembali 2 buah pasangan minterm yang memiliki 1 buah perbedaan yang kemudian akan saling dipasangkan dan hasilnya akan disimpan dalam sebuah tabel (tabel 2). Tabel yang diperoleh adalah sebagai berikut.

Group	Matched-Pairs	Representasi Binary (ABCD)
Group 0	m1-m3	00X1
Group 1	m3-m7	0X11
	m12-m13-m14-m15	11XX
	m12-m14-m13-m15	11XX
Group 2	m7-m15	X111

Tabel 2.3. Tabel Reduksi 2

4. Kemudian untuk pasangan minterm yang memiliki representasi biner sama akan direduksi sehingga diperoleh hasil akhir berupa tabel *prime implicant* yang menandakan bahwa tidak ada lagi pasangan minterm yang dapat direduksi. Tabel yang diperoleh adalah sebagai berikut.

Group	Matched-Pairs	Representasi Binary (ABCD)	Minimized Form
Group 0	m1-m3	00X1	A'B'D
Group 1	m3-m7	0X11	A'CD
Group 2	m7-m15	X111	BCD
Group 3	m12-m13-m14-m15	11XX	AB

Tabel 2.4. Tabel Prime Implicant (PI)

5. Kemudian dari tabel *prime implicant* tersebut akan dipilih set yang terdapat di dalam PI yang mengandung paling sedikit kemungkinan jumlah PI dan mencakup semua minterm yang terdapat dalam ekspresi fungsi *boolean algebra* sehingga diperoleh tabel *essential prime implicant* sebagai berikut.

<i>Prime Implicant</i>	<i>Minterm Involved</i>	<i>Minterms given in the problem</i>						
		1	3	7	12	13	14	15
A'B'D	(1,3)	X	X					
A'CD	(3,7)		X	X				
BCD	(7,15)			X				X
AB	(12,13,14,15)				X	X	X	X

Tabel 2.5. Tabel Essential Prime Implicant (EPI)

Kemudian dari tabel EPI tersebut diperoleh hasil akhir atau *output* berupa hasil minimisasi dari ekspresi fungsi logika awal, *output* yang diperoleh adalah sebagai berikut.

$$f(A, B, C, D) = A'B'D + A'CD + AB$$

2.2 Flowchart

Oleh karena algoritma yang kelompok kami gunakan adalah algoritma Quine-McCluskey, maka dapat diketahui bahwa *input* yang akan diberikan dari program simulasi yang akan kami buat adalah berupa informasi mengenai ekspresi fungsi *boolean algebra* yang ingin dicari ekspresi minimumnya.

2.2 Data Flow Diagram (DFD)

Oleh karena algoritma yang kelompok kami gunakan adalah algoritma Quine-McCluskey, maka dapat diketahui bahwa *input* yang akan diberikan dari program simulasi yang akan kami buat adalah berupa informasi mengenai ekspresi fungsi *boolean algebra* yang ingin dicari ekspresi minimumnya.

3. IMPLEMENTASI PROGRAM DALAM BAHASA C

3.1 Link GitHub Source Code

Berikut ini merupakan *link repository* GitHub dari kelompok kami.

<https://github.com/fitranurindra/Logic-Minimization>

Adapun penjelasan dari hierarki dan struktur *file* dalam *branch main* di *repository* GitHub tersebut adalah sebagai berikut.

1. B
2. C
3. D
4. E

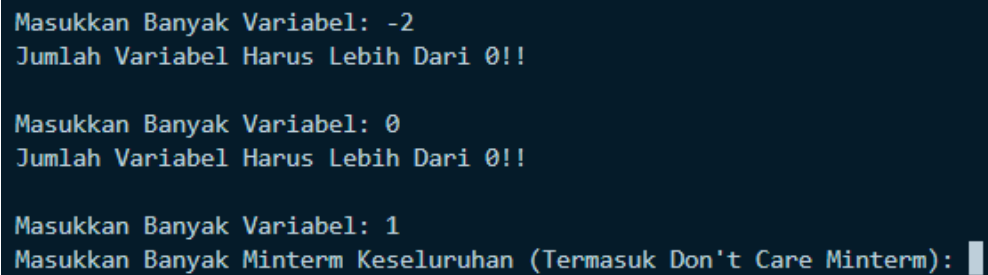
3.2 Simulasi Program

Untuk mengukur keakuratan program, dilakukan benchmarking atau perbandingan hasil minimisasi fungsi logika dari program yang telah dibuat dengan *logic minimization calculator* dari internet yaitu <https://www.charlie-coleman.com/experiments/kmap/>.

Simulasi program yang akan dilakukan akan dibagi menjadi beberapa test kasus (*case test*) sebagai berikut.

1. Kasus 1: *Input* jumlah variabel *invalid*

Pada kasus 1 ini, program akan diberikan *input* jumlah variabel yang *invalid*. Program akan terus melakukan validasi hingga *user* memberikan *input* jumlah variabel yang *valid*. *Input* jumlah variabel yang *valid* adalah harus lebih besar dari 0. Untuk *input* jumlah variabel yang *invalid* diperoleh hasil pengujian sebagai berikut.



```
Masukkan Banyak Variabel: -2
Jumlah Variabel Harus Lebih Dari 0!!

Masukkan Banyak Variabel: 0
Jumlah Variabel Harus Lebih Dari 0!!

Masukkan Banyak Variabel: 1
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm):
```

Gambar 3.1. Hasil Pengujian Kasus 1 (*Input* Jumlah Variabel *Invalid*)

Dari hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil sesuai dengan yang diinginkan bahwa skema validasi telah berjalan sebagaimana mestinya.

2. Kasus 2: *Input* banyak minterm keseluruhan *invalid*

Pada kasus 2 ini, program akan diberikan *input* banyak minterm keseluruhan *invalid*. Program akan terus melakukan validasi hingga *user* memberikan *input* banyak minterm keseluruhan yang *valid*. *Input* banyak minterm keseluruhan yang *valid* adalah harus lebih besar dari 0 dan kurang dari $2^{nVariable} + 1$. Untuk *input* banyak minterm keseluruhan yang *invalid* diperoleh hasil pengujian sebagai berikut.


```

Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): -1
Banyak Minterm Tidak Dapat Lebih Besar Dari 2^1 atau Lebih Kecil Dari 1!

Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 0
Banyak Minterm Tidak Dapat Lebih Besar Dari 2^1 atau Lebih Kecil Dari 1!

Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 3
Banyak Minterm Tidak Dapat Lebih Besar Dari 2^1 atau Lebih Kecil Dari 1!

Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 2
Masukkan Banyak Don't Care Minterm: 

```

Gambar 3.2. Hasil Pengujian Kasus 2 (*Input Banyak Minterm Keseluruhan Invalid*)

Dari hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil sesuai dengan yang diinginkan bahwa skema validasi telah berjalan sebagaimana mestinya.

3. Kasus 3: *Input banyak don't care minterm invalid*

Pada kasus 3 ini, program akan diberikan *input* banyak *don't care* minterm *invalid*. Program akan terus melakukan validasi hingga *user* memberikan *input* banyak *don't care* minterm yang *valid*. *Input* banyak *don't care* minterm yang *valid* adalah harus lebih besar dari sama dengan 0 dan kurang dari banyak minterm keseluruhan. Untuk *input* banyak *don't care* minterm yang *invalid* diperoleh hasil pengujian sebagai berikut.

```

Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 2
Masukkan Banyak Don't Care Minterm: 2
Banyak Don't Care Minterm Tidak Dapat Lebih Besar Dari Sama Dengan Banyak Minterm Keseluruhan atau Lebih Kecil Dari 0!

Masukkan Banyak Don't Care Minterm: 3
Banyak Don't Care Minterm Tidak Dapat Lebih Besar Dari Sama Dengan Banyak Minterm Keseluruhan atau Lebih Kecil Dari 0!

Masukkan Banyak Don't Care Minterm: -1
Banyak Don't Care Minterm Tidak Dapat Lebih Besar Dari Sama Dengan Banyak Minterm Keseluruhan atau Lebih Kecil Dari 0!

Masukkan Banyak Don't Care Minterm: 0

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!
Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 

```

Gambar 3.3. Hasil Pengujian Kasus 3 (*Input Banyak Don't Care Minterm Invalid*)

Dari hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil sesuai dengan yang diinginkan bahwa skema validasi telah berjalan sebagaimana mestinya.

4. Kasus 4: *Input indeks minterm invalid*

Pada kasus 4 ini, program akan diberikan *input* indeks minterm *invalid*. Program akan terus melakukan validasi hingga *user* memberikan *input* indeks minterm yang *valid*. *Input* indeks minterm yang *valid* adalah harus dimulai dari indeks terkecil hingga indeks terbesar (terurut membesar) serta *input* indeks minterm yang *valid* haruslah lebih besar dari sama dengan 0 dan kurang dari $2^{nVariable}$. Untuk *input* indeks minterm yang *invalid* diperoleh hasil pengujian sebagai berikut.

```

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 1
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 0
Input Indeks Minterm Tidak Dalam Urutan Meningkat
Masukkan Kembali Seluruh Indeks Minterm Dari Awal

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): -1

Indeks Minterm Harus Lebih Besar Dari Sama Dengan 0 dan Lebih Kecil Dari 2
Masukkan Kembali Seluruh Indeks Minterm Dari Awal

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 2

Indeks Minterm Harus Lebih Besar Dari Sama Dengan 0 dan Lebih Kecil Dari 2
Masukkan Kembali Seluruh Indeks Minterm Dari Awal

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 1

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

```

Gambar 3.4. Hasil Pengujian Kasus 4 (*Input Indeks Minterm Invalid*)

Dari hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil sesuai dengan yang diinginkan bahwa skema validasi telah berjalan sebagaimana mestinya.

5. Kasus 5: *Input indeks don't care minterm invalid*

Pada kasus 5 ini, program akan diberikan *input* indeks *don't care* minterm *invalid*. Program akan terus melakukan validasi hingga *user* memberikan *input* indeks *don't care* minterm yang *valid*. *Input* indeks *don't care* minterm yang *valid* adalah harus dimulai dari indeks terkecil hingga indeks terbesar (terurut membesar) serta *input* indeks *don't care* minterm yang *valid* haruslah lebih besar dari sama dengan 0 dan kurang dari $2^{nVariable}$. Untuk *input* indeks *don't care* minterm yang *invalid* diperoleh hasil pengujian sebagai berikut.

```

Masukkan Indeks Minterm yang Merupakan Don't Care Minterm!

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 2
Masukkan Indeks Don't Care Minterm ke-2 (Dalam Urutan Meningkat): 3
Masukkan Indeks Don't Care Minterm ke-3 (Dalam Urutan Meningkat): 1
Input Indeks Don't Care Minterm Tidak Dalam Urutan Meningkat
Masukkan Kembali Seluruh Indeks Don't Care Minterm Dari Awal

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): -1

Indeks Don't Care Minterm Harus Lebih Besar Dari Sama Dengan 0 dan Lebih Kecil Dari 8
Masukkan Kembali Seluruh Indeks Don't Care Minterm Dari Awal

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 8

Indeks Don't Care Minterm Harus Lebih Besar Dari Sama Dengan 0 dan Lebih Kecil Dari 8
Masukkan Kembali Seluruh Indeks Don't Care Minterm Dari Awal

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Don't Care Minterm ke-2 (Dalam Urutan Meningkat): 1
Masukkan Indeks Don't Care Minterm ke-3 (Dalam Urutan Meningkat): 2

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

```

Gambar 3.5. Hasil Pengujian Kasus 5 (*Input Indeks Don't Care Minterm Invalid*)

Dari hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil sesuai dengan yang diinginkan bahwa skema validasi telah berjalan sebagaimana mestinya.

6. Kasus 6: Kasus *input* 1 variabel

Pada kasus 6 ini, program akan diberikan *input* 1 variabel. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A) = \sum m(0)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 1
- Banyak minterm keseluruhan = 1
- Banyak *don't care* minterm = 0
- Indeks minterm keseluruhan = 0

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```
Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 1
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 1
Masukkan Banyak Don't Care Minterm: 0

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!:

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

A'

Tekan Tombol Apapun Untuk Keluar!
```

Gambar 3.6. Hasil Pengujian Kasus 6 (*Input* 1 Variabel)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: One string for function result f	Minterms: Comma separated list of numbers 0	Generic: f(A) = A'
Input Names: Comma separated list of variable names A	Don't Cares: Comma separated list of numbers	VHDL:

Gambar 3.7. Hasil Pengujian Kasus 6 (*Input* 1 Variabel Dengan *Logic Minimization Calculator* Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 1 variabel.

7. Kasus 7: Kasus *input* 2 variabel tanpa *don't care*

Pada kasus 7 ini, program akan diberikan *input* 2 variabel tanpa *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B) = \sum m(0, 2)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 2
- Banyak minterm keseluruhan = 2
- Banyak *don't care* minterm = 0
- Indeks minterm keseluruhan = 0, 2

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 2
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 2
Masukkan Banyak Don't Care Minterm: 0

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 2

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

B'

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.8. Hasil Pengujian Kasus 7 (Input 2 Variabel Tanpa Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: <small>One string for function result</small> f	Minterms: <small>Comma separated list of numbers</small> 0,2	Generic: f(A, B) = B'
Input Names: <small>Comma separated list of variable names</small> A, B	Don't Cares: <small>Comma separated list of numbers</small>	VHDL:

Gambar 3.9. Hasil Pengujian Kasus 7 (Input 2 Variabel Tanpa Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input 2* variabel tanpa *don't care*.

8. Kasus 8: Kasus *input 2* variabel dengan *don't care*
 Pada kasus 7 ini, program akan diberikan *input 2* variabel dengan *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B) = \sum m(0,3) + \sum d(2)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 2
- Banyak minterm keseluruhan = 3
- Banyak *don't care* minterm = 1
- Indeks minterm keseluruhan = 0, 2, 3
- Indeks *don't care* minterm = 2

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 2
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 3
Masukkan Banyak Don't Care Minterm: 1

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 2
Masukkan Indeks Minterm ke-3 (Dalam Urutan Meningkat): 3

Masukkan Indeks Minterm yang Merupakan Don't Care Minterm!

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 2

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

B' + A

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.10. Hasil Pengujian Kasus 8 (Input 2 Variabel Dengan Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: One string for function result f	Minterms: Comma separated list of numbers 0,3	Generic: f(A, B) = B' + A
Input Names: Comma separated list of variable names A, B	Don't Cares: Comma separated list of numbers 2	VHDL:

Gambar 3.11. Hasil Pengujian Kasus 8 (Input 2 Variabel Dengan Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 2 variabel dengan *don't care*.

9. Kasus 9: Kasus *input* 3 variabel tanpa *don't care*

Pada kasus 9 ini, program akan diberikan *input* 3 variabel tanpa *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B, C) = \sum m(0, 1, 4, 6)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 3
- Banyak minterm keseluruhan = 4
- Banyak *don't care* minterm = 0
- Indeks minterm keseluruhan = 0, 1, 4, 6

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 3
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 4
Masukkan Banyak Don't Care Minterm: 0

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 1
Masukkan Indeks Minterm ke-3 (Dalam Urutan Meningkat): 4
Masukkan Indeks Minterm ke-4 (Dalam Urutan Meningkat): 6

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

A'B' + AC'

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.12. Hasil Pengujian Kasus 9 (Input 3 Variabel Tanpa Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: One string for function result f	Minterms: Comma separated list of numbers 0,1,4,6	Generic: f(A, B, C) = A'B' + AC'
Input Names: Comma separated list of variable names A, B, C	Don't Cares: Comma separated list of numbers	VHDL:

Gambar 3.13. Hasil Pengujian Kasus 9 (Input 3 Variabel Tanpa Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 3 variabel tanpa *don't care*.

10. Kasus 10: Kasus *input* 3 variabel dengan *don't care*

Pada kasus 10 ini, program akan diberikan *input* 3 variabel dengan *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B, C) = \sum m(0, 1, 4, 7) + \sum d(3)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 3
- Banyak minterm keseluruhan = 5
- Banyak *don't care* minterm = 1
- Indeks minterm keseluruhan = 0, 1, 3, 4, 7
- Indeks *don't care* minterm = 3

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 3
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 5
Masukkan Banyak Don't Care Minterm: 1

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 1
Masukkan Indeks Minterm ke-3 (Dalam Urutan Meningkat): 3
Masukkan Indeks Minterm ke-4 (Dalam Urutan Meningkat): 4
Masukkan Indeks Minterm ke-5 (Dalam Urutan Meningkat): 7

Masukkan Indeks Minterm yang Merupakan Don't Care Minterm!

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 3

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

B'C' + BC + A'B'

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.14. Hasil Pengujian Kasus 10 (Input 3 Variabel Dengan Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: One string for function result f	Minterms: Comma separated list of numbers 0,1,4,7	Generic: f(A, B, C) = B'C' + BC + A'B'
Input Names: Comma separated list of variable names A, B, C	Don't Cares: Comma separated list of numbers 3	VHDL:

Gambar 3.15. Hasil Pengujian Kasus 10 (Input 3 Variabel Dengan Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 3 variabel dengan *don't care*.

11. Kasus 11: Kasus *input* 4 variabel tanpa *don't care*

Pada kasus 11 ini, program akan diberikan *input* 4 variabel tanpa *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B, C, D) = \sum m(1, 3, 7, 12, 13, 14, 15)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 4
- Banyak minterm keseluruhan = 7
- Banyak *don't care* minterm = 0
- Indeks minterm keseluruhan = 1, 3, 7, 12, 13, 14, 15

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.

```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 4
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 7
Masukkan Banyak Don't Care Minterm: 0

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 1
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 3
Masukkan Indeks Minterm ke-3 (Dalam Urutan Meningkat): 7
Masukkan Indeks Minterm ke-4 (Dalam Urutan Meningkat): 12
Masukkan Indeks Minterm ke-5 (Dalam Urutan Meningkat): 13
Masukkan Indeks Minterm ke-6 (Dalam Urutan Meningkat): 14
Masukkan Indeks Minterm ke-7 (Dalam Urutan Meningkat): 15

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

A'B'D + A'CD + AB

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.16. Hasil Pengujian Kasus 11 (Input 4 Variabel Tanpa Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: One string for function result f	Minterms: Comma separated list of numbers 1,3,7,12,13,14,15	Generic: f(A, B, C, D) = A'B'D + AB + A'CD
Input Names: Comma separated list of variable names A, B, C, D	Don't Cares: Comma separated list of numbers	VHDL:

Gambar 3.17. Hasil Pengujian Kasus 11 (Input 4 Variabel Tanpa Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 4 variabel tanpa *don't care*.

12. Kasus 12: Kasus *input* 4 variabel dengan *don't care*

Pada kasus 11 ini, program akan diberikan *input* 4 variabel dengan *don't care*. Program akan melakukan minimisasi dari ekspresi fungsi logika berikut.

$$f(A, B, C, D) = \sum m(0, 3, 10, 15) + \sum d(1, 2, 7, 8, 11, 14)$$

Sehingga *inputnya* adalah:

- Banyak variabel = 4
- Banyak minterm keseluruhan = 10
- Banyak *don't care* minterm = 6
- Indeks minterm keseluruhan = 0, 1, 2, 3, 7, 8, 10, 11, 14, 15
- Indeks *don't care* minterm = 1, 2, 7, 8, 11, 14

Untuk kasus ini diperoleh hasil pengujian sebagai berikut.


```

Selamat Datang di Program Minimisasi Logic.
Silakan Masukkan Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi

Masukkan Banyak Variabel: 4
Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care Minterm): 10
Masukkan Banyak Don't Care Minterm: 6

Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!

Masukkan Indeks Minterm ke-1 (Dalam Urutan Meningkat): 0
Masukkan Indeks Minterm ke-2 (Dalam Urutan Meningkat): 1
Masukkan Indeks Minterm ke-3 (Dalam Urutan Meningkat): 2
Masukkan Indeks Minterm ke-4 (Dalam Urutan Meningkat): 3
Masukkan Indeks Minterm ke-5 (Dalam Urutan Meningkat): 7
Masukkan Indeks Minterm ke-6 (Dalam Urutan Meningkat): 8
Masukkan Indeks Minterm ke-7 (Dalam Urutan Meningkat): 10
Masukkan Indeks Minterm ke-8 (Dalam Urutan Meningkat): 11
Masukkan Indeks Minterm ke-9 (Dalam Urutan Meningkat): 14
Masukkan Indeks Minterm ke-10 (Dalam Urutan Meningkat): 15

Masukkan Indeks Minterm yang Merupakan Don't Care Minterm!

Masukkan Indeks Don't Care Minterm ke-1 (Dalam Urutan Meningkat): 1
Masukkan Indeks Don't Care Minterm ke-2 (Dalam Urutan Meningkat): 2
Masukkan Indeks Don't Care Minterm ke-3 (Dalam Urutan Meningkat): 7
Masukkan Indeks Don't Care Minterm ke-4 (Dalam Urutan Meningkat): 8
Masukkan Indeks Don't Care Minterm ke-5 (Dalam Urutan Meningkat): 11
Masukkan Indeks Don't Care Minterm ke-6 (Dalam Urutan Meningkat): 14

Fungsi Logika Setelah Minimisasi Dalam Bentuk SOP:

A'B' + AC

Tekan Tombol Apapun Untuk Keluar!

```

Gambar 3.18. Hasil Pengujian Kasus 12 (Input 4 Variabel Dengan Don't Care)

Ketika dilakukan pengujian dengan menggunakan *logic minimization calculator* dari internet diperoleh hasil sebagai berikut.

Function Info	Terms	Solutions:
Output Name: <small>One string for function result</small> f	Minterms: <small>Comma separated list of numbers</small> 0,3,10,15	Generic: f(A, B, C, D) = A'B' + AC
Input Names: <small>Comma separated list of variable names</small> A, B, C, D	Don't Cares: <small>Comma separated list of numbers</small> 1,2,7,8,11,14	VHDL:

Gambar 3.19. Hasil Pengujian Kasus 12 (Input 4 Variabel Dengan Don't Care Dengan Logic Minimization Calculator Dari Internet)

Dari kedua hasil pengujian tersebut dapat diketahui bahwa telah diperoleh hasil yang sama dan sesuai dengan yang diharapkan, sehingga simulasi program telah sebagaimana mestinya untuk kasus *input* 4 variabel dengan *don't care*.

3.3 Source Code

3.3.1 Source Code minimisasi_lib.h

```
/** EL2008 Pemecahan Masalah dengan C 2020/2021
 * Tugas Besar      : Minimisasi Logic
 * Kelompok         : 6
 * Nama File        : minimisasi_lib.h
 * Deskripsi        : Header file untuk library minimisasi.
 */

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

/** fillAllMT. Prosedur untuk mengisi nilai dari array IdxMinterm yang
berisi seluruh indeks minterm yang ingin diminimisasi.
 *
 * @param nAllMT merupakan banyaknya minterm secara keseluruhan (termasuk
don't care minterm).
 * @param nVariable merupakan banyaknya variabel dalam fungsi logika yang
ingin diminimisasi.
 * @param IdxMinterm merupakan array of integer yang berisi seluruh
indeks minterm yang akan diminimisasi.
 *
 */
void fillAllMT(int nAllMT, int nVariable, int *IdxMinterm);

/** fillDontCareMT. Prosedur untuk mengisi nilai dari array IdxMintermDC
yang berisi seluruh indeks don't care minterm yang ingin diminimisasi.
 *
 * @param nDontCareMT merupakan banyaknya don't care minterm.
 * @param nVariable merupakan banyaknya variabel dalam fungsi logika yang
ingin diminimisasi.
 * @param IdxMintermDC merupakan array of integer yang berisi seluruh
indeks don't care minterm yang akan diminimisasi.
 *
 */
void fillDontCareMT(int nDontCareMT, int nVariable, int *IdxMintermDC);

/** DecToBin. Prosedur untuk mengubah indeks minterm yang berformat
desimal menjadi indeks minterm dalam format binary.
 *
 * @param nAllMT merupakan banyaknya minterm secara keseluruhan (termasuk
don't care minterm).
 * @param nVariable merupakan banyaknya variabel dalam fungsi logika yang
ingin diminimisasi.
 * @param IdxMinterm merupakan array of integer yang berisi seluruh
indeks minterm yang akan diminimisasi.
 * @param MTinBinary merupakan array of array integer yang berisi seluruh
indeks minterm yang akan diminimisasi dalam format binary.
 *
 */
void DecToBin(int nAllMT, int nVariable, int *IdxMinterm, int
**MTinBinary);

/** countOne. Fungsi untuk menghitung banyaknya angka 1 yang terdapat
pada array of integer yang merepresentasikan bilangan dalam bentuk biner.
 *
 * @param binary merupakan array of integer yang merepresentasikan
bilangan dalam bentuk biner.
```

```

* @param nDigit merupakan banyaknya digit pada bilangan biner (panjang
array of integer).
* @return count merupakan banyaknya angka 1 yang terdapat pada array of
integer yang merepresentasikan bilangan dalam bentuk biner.
*
*/
int countOne (int *binary, int nDigit);

/** Combination. Fungsi untuk menghitung banyaknya kombinasi berdasarkan
kolom yang diberikan.
*
* @param n merupakan banyaknya bilangan secara keseluruhan yang akan
dihitung kombinasinya (dalam kasus ini berupa banyaknya variabel yang
diberikan).
* @param ColumnNo merupakan nomor kolom yang akan menentukan banyaknya
kombinasi.
* @param k merupakan kondisi lain dalam setiap nomor kolom yang sama.
* @return Comb merupakan banyaknya kombinasi yang dihasilkan berdasarkan
input yang telah diberikan.
*
*/
int Combination(int n, int ColumnNo, int k);

/** IsPowerOfTwo. Fungsi untuk mengecek apakah suatu bilangan dapat
direpresentasikan ke dalam bentuk  $2^n$ .
*
* @param n merupakan bilangan yang akan dicek.
* @return (floor(log(n)/log(2)) == (log(n)/log(2))) merupakan kondisi
True/False, jika True akan memberikan hasil 1, sedangkan jika False akan
memberikan hasil 0.
*
*/
int IsPowerOfTwo(int n);

/** Grouping. Prosedur untuk melakukan proses grouping.
*
* @param nVariable merupakan banyaknya variabel dalam fungsi logika yang
ingin diminimisasi.
* @param Column merupakan column yang menyimpan hasil dari setiap proses
grouping.
*
*/
void Grouping(int nVariable, int ****Column);

/** recursive. Prosedur untuk menyimpan seluruh nilai prime implicant
yang memenuhi kriteria EPI dengan cara rekursif.
*
* @param m merupakan
* @param nSisaMT merupakan banyaknya minterm yang masih tersisa atau
belum tercover dalam IdxEPI.
* @param nSisaPI merupakan banyaknya prime implicant yang masih tersisa
atau belum tercover dalam IdxEPI.
* @param reduksiPChart merupakan array of array integer yang berisi
prime implicant hasil reduksi (yang masih tersisa).
* @param chooseEPI merupakan array of integer untuk menyimpan nilai
prime implicant yang memenuhi kriteria EPI.
* @param potentialEPI merupakan array of array integer untuk menyimpan
seluruh nilai prime implicant yang memenuhi kriteria EPI.
*
*/

```

```

void recursive(int m, int nSisaMT, int nSisaPI, int **reduksiPIChart, int
*chooseEPI, int **potentialEPI);

/** minResult. Prosedur untuk menampilkan hasil akhir dari proses
minimisasi fungsi logika.
*
* @param nVariable merupakan banyaknya variabel dalam fungsi logika yang
ingin diminimisasi.
* @param nEPI merupakan jumlah minimum prime implicant yang sudah
mengcover seluruh minterm.
* @param IdxEPI merupakan array of array integer yang berisi prime
implicant dalam jumlah minimum namun sudah mengcover seluruh minterm.
* @param Column merupakan column yang menyimpan hasil dari setiap proses
grouping
*
*/
void minResult(int nVariable, int nEPI, int **IdxEPI, int ****Column);

```

3.3.2 Source Code minimisasi_lib.c

```
/** EL2008 Pemecahan Masalah dengan C 2020/2021
 * Tugas Besar      : Minimisasi Logic
 * Kelompok        : 6
 * Nama File       : minimisasi_lib.c
 * Deskripsi       : Implementation file untuk library minimisasi.
 */

#include "minimisasi_lib.h"

void fillAllMT(int nAllMT, int nVariable, int *IdxMinterm){
    int i;

    for(i = 0; i < nAllMT; i++){
        printf("Masukkan Indeks Minterm ke-%d (Dalam Urutan Meningkat):", i+1);
        scanf("%d", &IdxMinterm[i]);

        // Memvalidasi Input Indeks Minterm
        if(i != 0 && IdxMinterm[i] <= IdxMinterm[i-1]){
            printf("Input Indeks Minterm Tidak Dalam Urutan Meningkat\n");
            printf("Masukkan Kembali Seluruh Indeks Minterm Dari Awal\n\n");
            i=-1;
        }
        else if(IdxMinterm[i] >= pow(2, nVariable) || IdxMinterm[i] < 0){
            int a = pow(2, nVariable);
            printf("\nIndeks Minterm Harus Lebih Besar Dari Sama Dengan 0 dan Lebih Kecil Dari %d\n", a);
            printf("Masukkan Kembali Seluruh Indeks Minterm Dari Awal\n\n");
            i=-1;
        }
    }
}

void fillDontCareMT(int nDontCareMT, int nVariable, int *IdxMintermDC){
    int i;

    if(nDontCareMT != 0){
        printf("\nMasukkan Indeks Minterm yang Merupakan Don't Care Minterm! \n\n");

        // Mengisi Nilai Array IdxMintermD dan Memvalidasinya
        for(i=0; i < nDontCareMT; i++){
            printf("Masukkan Indeks Don't Care Minterm ke-%d (Dalam Urutan Meningkat): ", i+1);
            scanf("%d", &IdxMintermDC[i]);

            // Memvalidasi Input Indeks Don't Care Minterm
            if(i!=0 && IdxMintermDC[i] <= IdxMintermDC[i-1]){
                printf("Input Indeks Don't Care Minterm Tidak Dalam Urutan Meningkat\n");
                printf("Masukkan Kembali Seluruh Indeks Don't Care Minterm Dari Awal\n\n");
                i=-1;
            }
        }
    }
}
```

```

        else if (IdxMintermDC[i] >= pow(2, nVariable) || IdxMintermDC[i]
< 0){
            int a = pow(2, nVariable);
            printf("\nIndeks Don't Care Minterm Harus Lebih Besar
Dari Sama Dengan 0 dan Lebih Kecil Dari %d\n", a);
            printf("Masukkan Kembali Seluruh Indeks Don't Care
Minterm Dari Awal\n\n");
            i = -1;
        }
    }
}

void DecToBin(int nAllMT, int nVariable, int *IdxMinterm, int
**MTinBinary){
    int i, j, temp;

    for(i = 0; i < nAllMT; i++){
        temp = IdxMinterm[i];

        for(j = nVariable-1; j >= 0; j--){
            MTinBinary[i][j] = temp%2;
            temp = temp/2;
        }
    }
}

int countOne (int *binary, int nDigit){
    int i, count = 0;

    for(i = 0; i <= nDigit-1; i++){
        if(binary[i] == 1){
            count++;
        }
    }

    return count;
}

int Combination(int n, int ColumnNo, int k){
    int Comb, i, NtoK = 1, Kto1 = 1;

    for(i = n; i >= n-k+1-ColumnNo; i--){
        NtoK = i*NtoK;
    }

    for(i = k; i >= 1; i--){
        Kto1 = i * Kto1;
    }

    Comb = NtoK/Kto1;

    return Comb;
}

int IsPowerOfTwo(int n){
    return (floor(log(n)/log(2)) == (log(n)/log(2)));
}

void Grouping(int nVariable, int ****Column){
    int i, j, k, l, m, n, p;

```

```

int logicCheck, position;
int groupAble = 1; // Inisialisasi Kondisi Awal

for(i = 0; i < nVariable + 1; i++){
    if(groupAble != 0){
        groupAble = 0;

        for(j = 0; j < nVariable - i; j++){
            m = 0;

            for(k = 0; k < Combination(nVariable,i,j); k++){
                if(Column[i][j][k] != NULL){
                    for(l = 0; l < Combination(nVariable,i,j+1);
l++){
                        if(Column[i][j+1][l] != NULL &&
Column[i][j+1][l][nVariable+2+i] > Column[i][j][k][nVariable+2+i]){

if(IsPowerOfTwo(Column[i][j+1][l][nVariable+2+i]-
Column[i][j][k][nVariable+2+i])){
                                logicCheck = 0 - i; // Digunakan
untuk mengecek apakah 2 minterm terletak pada posisi yang sama
(direpresentasikan dengan 2)

                                for(n = 1; n <= i; n++){
                                    for(p = 1; p <= i; p++){

if(Column[i][j+1][l][nVariable+1+n] == Column[i][j][k][nVariable+1+p]){
                                        logicCheck++;
                                    }
                                }
                            }

                            if(logicCheck == 0){
                                groupAble = 1;
                                Column[i][j][k][nVariable+1] = 1;
                                Column[i][j+1][l][nVariable+1] =
1;

                                Column[i+1][j][m] = (int
*)malloc((nVariable+4+i*pow(2,i+1)) * sizeof(int));

                                for(n = 0; n <= nVariable+1+i;
n++){
                                    Column[i+1][j][m][n] =
Column[i][j][k][n];

                                    Column[i+1][j][m][nVariable+3+i]
= Column[i][j][k][nVariable+2+i];

                                    for(n = nVariable+4+i; n <
nVariable+4+i*pow(2,i+1); n++){
                                        Column[i+1][j][m][n] = 0;
                                    }

                                    position =
log((Column[i][j+1][l][nVariable+2+i]-
Column[i][j][k][nVariable+2+i]))/log(2);

                                    Column[i+1][j][m][nVariable-1-
position] = 2; // Terletak Pada Posisi yang Sama

```

```

Column[i+1][j][m][nVariable+1] =
0;
Column[i+1][j][m][nVariable+2+i]
= position;

for(p = 0; p < pow(2,i); p++){
Column[i+1][j][m][nVariable+4+i+p] = Column[i][j][k][nVariable+3+i+p];
}

for(p = pow(2,i); p < pow(2,i+1);
p++){
Column[i+1][j][m][nVariable+4+i+p] = Column[i][j+1][l][nVariable+3+i+p-
(int)pow(2,i)];
}
m++;
}
}
}
}
}
}
}
}
}

void recursive(int m, int nSisaMT, int nSisaPI, int **reduksiPICChart, int
*chooseEPI, int **potentialEPI){
int i;
int n = m;
int PotEPINo = 0;

for(chooseEPI[n] = 0; chooseEPI[n] < nSisaPI; chooseEPI[n]++){
if(reduksiPICChart[nSisaMT-1-n][chooseEPI[n]]){
if(n > 0){
m = n;
m--;
recursive(m, nSisaMT, nSisaPI, reduksiPICChart, chooseEPI,
potentialEPI);
}

else if(n == 0){
for(i = 0; i < nSisaMT;i++){
potentialEPI[PotEPINo][i] = chooseEPI[nSisaMT-1-i];
}

PotEPINo++;
}
}
}
}

void minResult(int nVariable, int nEPI, int **IdxEPI, int ****Column){
printf("\nFungsi Logika Setelah Minimisasi Dalam Bentuk SOP:\n\n");
printf("\n ");

```



```

int x, y;
for(x = 0; x < nEPI; x++){
    for(y = 0; y < nVariable; y++){
        if(Column[IdxEPI[x][0]][IdxEPI[x][1]][IdxEPI[x][2]][y] == 1){
            printf("%c", 65 + y);
        }

        else if(Column[IdxEPI[x][0]][IdxEPI[x][1]][IdxEPI[x][2]][y]
== 0){
            printf("%c", 65 + y);
        }

    }

    if(x < nEPI-1){
        printf(" + ");
    }

}

printf("\n\nTekan Tombol Apapun Untuk Keluar!\n\n");

getch();
}

```

3.3.3 Source Code main.c

```

/** EL2008 Pemecahan Masalah dengan C 2020/2021
 * Tugas Besar      : Minimisasi Logic
 * Kelompok         : 6
 * Nama File        : main.c
 * Deskripsi        : Program utama untuk meminimisasi logic dengan
menggunakan algoritma
 * Quine McCluskey (Metode Tabular) berdasarkan input yang diberikan oleh
user dan menampilkan hasilnya pada layar.
 */

#include "minimisasi_lib.c"

int main(){
    int nVariable, nAllMT, nDontCareMT;
    int i, j, k;

    printf("Selamat Datang di Program Minimisasi Logic.\nSilakan Masukkan
Informasi Mengenai Ekspresi Logic yang Ingin Diminimisasi\n\n");

    //--Algoritma Untuk Menyimpan Input Dari User Serta Memvalidasinya--
    //
    printf("Masukkan Banyak Variabel: ");
    scanf("%d",&nVariable);

    // Memvalidasi Input Jumlah Variabel
    while(nVariable<=0){
        printf("Jumlah Variabel Harus Lebih Dari 0!!\n\n");
        printf("Masukkan Banyak Variabel: ");
        scanf("%d",&nVariable);
    }
}

```

```

printf("Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care
Minterm): ");
scanf("%d",&nAllMT);

// Memvalidasi Input Banyak Minterm Keseluruhan
while(nAllMT>pow(2,nVariable) || nAllMT<=0){
    printf("Banyak Minterm Tidak Dapat Lebih Besar Dari 2^%d atau
Lebih Kecil Dari 1!\n\n",nVariable);
    printf("Masukkan Banyak Minterm Keseluruhan (Termasuk Don't Care
Minterm): ");
    scanf("%d",&nAllMT);
}

printf("Masukkan Banyak Don't Care Minterm: ");
scanf("%d",&nDontCareMT);

// Memvalidasi Input Banyak Don't Care Minterm
while(nDontCareMT >= nAllMT || nDontCareMT<0){
    printf("Banyak Don't Care Minterm Tidak Dapat Lebih Besar Dari
Sama Dengan Banyak Minterm Keseluruhan atau Lebih Kecil Dari 0!\n\n");
    printf("Masukkan Banyak Don't Care Minterm: ");
    scanf("%d",&nDontCareMT);
}

printf("\n");

// Array of Int Untuk Menyimpan Seluruh Indeks Minterm Dalam Desimal
int *IdxMinterm;
IdxMinterm=(int *)malloc(nAllMT*sizeof(int));

printf("Masukkan Minterm Keseluruhan (Termasuk Don't Care Minterm)!
\n\n");

// Mengisi Nilai Array IdxMinterm dan Memvalidasinya
fillAllMT(nAllMT, nVariable, IdxMinterm);

// Array of Int Untuk Menyimpan Seluruh Indeks Don't Care Minterm
Dalam Desimal
int *IdxMintermDC;
IdxMintermDC=(int *)malloc(nDontCareMT*sizeof(int));

// Mengisi Nilai Array IdxMintermDC dan Memvalidasinya Jika Terdapat
Don't Care Minterm
fillDontCareMT(nDontCareMT, nVariable, IdxMintermDC);

//--Algoritma Untuk Mengubah Indeks Desimal Minterm Menjadi Binary--
//
int **MTinBinary;

// Alokasi Memori Untuk Variabel MTinBinary
MTinBinary=(int **)malloc(nAllMT*sizeof(int*));

for(i = 0; i <= nAllMT; i++){
    MTinBinary[i]=(int *)malloc((nVariable+4)*sizeof(int));
}

// Mengubah Desimal Menjadi Binary
DecToBin(nAllMT, nVariable, IdxMinterm, MTinBinary);

// Inisialisasi Nilai Dari Matriks MTinBinary
for(i = 0; i < nAllMT; i++){

```

```

    MTinBinary[i][nVariable] = countOne(MTinBinary[i], nVariable); //
Menghitung Banyaknya Angka 1 Dari Seluruh Minterm

    MTinBinary[i][nVariable + 1] = 0; // Inisialisasi Kondisi Awal
Sebelum Dibentuk Grup Dengan Minterm Lain (0 Belum Dibentuk Grup, 1 Sudah
Dibentuk Grup Dengan Minterm Lain)

    MTinBinary[i][nVariable + 2] = IdxMinterm[i]; // Menyimpan Indeks
Minterm Dalam Bentuk Desimal

    MTinBinary[i][nVariable + 3] = IdxMinterm[i]; // Menyimpan Indeks
Minterm Keseluruhan Setelah Dibentuk Grup
}

//---Mempersiapkan Kolom Awal Untuk Grouping---//
int ****Column;

// Alokasi Memory Serta Inisialisasi Nilai Variabel Column
Column = (int ****)malloc((nVariable + 1) * sizeof(int***));

for(i = 0; i < nVariable + 1; i++){
    Column[i] = (int ***)malloc((nVariable + 1 - i) * sizeof(int**));
// Column[i] Untuk Menyimpan Grup Minterm Pada Kolom Ke-(i+1)
}

for(i = 0; i < nVariable + 1; i++){
    for(j = 0; j < nVariable + 1 - i; j++){
        Column[i][j] = (int**)malloc(Combination(nVariable,i,j) *
sizeof(int*)); // Column[i][j] Menyimpan Seluruh Minterm Dalam Bentuk
Binary Pada Kolom Ke-(i+1)
        for(k = 0; k < Combination(nVariable,i,j); k++){
            Column[i][j][k] = NULL; // Column[i][j][k] Representasi
Minterm Dalam Bentuk Binary Pada Kolom Ke-(i+1)
        }
    }
}

for(i = 0; i < nVariable + 1; i++){
    // i adalah
banyaknya grup sesuai banyaknya angka 1 : grup 1 (0 buah angka 1), grup 2
(1 buah angka 1), dst
    for(j = 0, k = 0; j < nAllMT; j++){
        if(MTinBinary[j][nVariable] == i){ // Membagi Grup
Sesuai Dengan Banyaknya Angka 1
            Column[0][i][k++] = MTinBinary[j]; // Column 0
(Berisi Minterm yang Telah Digrup Berdasarkan Banyaknya Angka 1)
        }
    }
}

//---Algoritma Untuk Proses Grouping Pada Kolom Selanjutnya---//

// Proses Membuat Group Dengan Metode Tabular (Quine-McCluskey)
Grouping(nVariable, Column);

//---Menghitung Seberapa Banyak Setiap Indeks Desimal Muncul---//
int *DecIdxCount;

// Alokasi Memory Serta Inisialisasi Nilai Variabel DecIdxCount
DecIdxCount=(int *)malloc(pow(2,nVariable) * sizeof(int));
for(i = 0; i < pow(2,nVariable); i++){
    DecIdxCount[i] = 0; // Inisialisasi Nilai Awal

```

```

    }

    //---Menyimpan Indeks Prime Implicant (Jika Terdapat Duplikat Akan
    Dihapus)---//
    int **IdxPI;

    // Alokasi Memory Variabel IdxPI
    IdxPI = (int **)malloc(nAllMT * sizeof(int*));
    for(i = 0; i < nAllMT; i++){
        IdxPI[i] = (int*)malloc(3 * sizeof(int));
    }

    int nPI = 0; // Inisialisasi Kondisi Awal
    int logicCheck;
    int l, m, n;

    // Mengisi Nilai Matriks/Tabel IdxPI
    for(i = 0; i < nVariable+1; i++){
        for(j = 0; j < nVariable+1-i; j++){
            for(k = 0; k < Combination(nVariable,i,j); k++){
                if(Column[i][j][k] != NULL &&
                Column[i][j][k][nVariable+1] == 0){
                    logicCheck = 0 - pow(2,i); // Untuk Mengecek Apakah
                    PI Duplikat Atau Tidak

                    for(l = k-1; l >= 0; l--){
                        if(logicCheck){
                            logicCheck = 0 - pow(2,i);

                            for(m = 0; m < pow(2,i); m++){
                                for(n = 0; n < pow(2,i); n++){
                                    if(Column[i][j][l][nVariable+3+i+m]
                                    == Column[i][j][k][nVariable+3+i+n]){
                                        logicCheck++;
                                    }
                                }
                            }
                        }
                    }

                    if(logicCheck != 0){
                        IdxPI[nPI][0] = i;
                        IdxPI[nPI][1] = j;
                        IdxPI[nPI][2] = k;

                        nPI++;

                        for(l = 0; l < pow(2,i); l++){
                            DecIdxCount[Column[i][j][k][nVariable+3+i+l]]++;
                        }
                    }
                }
            }
        }
    }

    //---Menghapus Don't Care Minterm yang Tidak Digunakan---//
    for(i = 0; i < nDontCareMT; i++){
        DecIdxCount[IdxMintermDC[i]] = 0;
    }

```

```

    }

    int **IdxEPI;

    // Alokasi Memory Variabel IdxEPI
    IdxEPI=(int **)malloc(nAllMT * sizeof(int*));

    int nEPI = 0; // Inisialisasi Nilai Awal

    //--Mengambil Minterm yang Hanya Muncul Satu Kali Pada Tabel PI dan
    Menyimpannya Pada EPI. Kemudian Mengeset Ulang DecIdxCountnya Menjadi 0--
    --//
    for(i = 0; i < pow(2,nVariable); i++){
        if(DecIdxCount[i] == 1){
            for(j = 0; j < nPI; j++){
                for(k = 0; k < pow(2,IdxPI[j][0]); k++){
                    if(Column[IdxPI[j][0]][IdxPI[j][1]][IdxPI[j][2]][nVariable+3+IdxPI[j][0]+
                    k] == i){
                        IdxEPI[nEPI] = IdxPI[j];

                        for(l = 0; l < pow(2,IdxPI[j][0]); l++){
                            DecIdxCount[Column[IdxPI[j][0]][IdxPI[j][1]][IdxPI[j][2]][nVariable+3+Idx
                            PI[j][0]+l]] = 0;
                        }

                        nEPI++;

                        k = pow(2,IdxPI[j][0]);
                    }
                }
            }
        }
    }

    //--Membuat Tabel Prime Implicant yang Sudah Tereduksi--//
    int nSisaMT = 0;

    for(i = 0; i < pow(2,nVariable); i++){
        if(DecIdxCount[i] != 0){
            nSisaMT++;
        }
    }

    int *reduksiPIChartX;

    // Alokasi Memory Serta Inisialisasi Nilai Variabel reduksiPIChartX
    reduksiPIChartX = (int *)malloc(nSisaMT * sizeof(int));
    for(i = 0; i < nSisaMT; i++){
        reduksiPIChartX[i] = -1;
    }

    int **reduksiPIChartY;

    // Alokasi Memory Serta Inisialisasi Nilai Variabel reduksiPIChartY
    reduksiPIChartY = (int **)malloc(nPI * sizeof(int*));
    for(i = 0; i < nPI; i++){
        reduksiPIChartY[i] = NULL;
    }

```

```

int **reduksiPIChart;

// Alokasi Memory Variabel reduksiPIChart
reduksiPIChart = (int **)malloc(nSisaMT * sizeof(int*));

//---Baris Pertama yang Terdiri Atas Minterm yang Masih Tersisa---//
for(i = 0, j = 0; j < pow(2,nVariable); j++){
    if(DecIdxCount[j] != 0)
    {
        reduksiPIChartX[i] = j;
        i++;
    }
}

//---Kolom Pertama yang Terdiri Atas Prime Implicant yang Masih
Tersisa---//
int nSisaPI = 0;

for(i = 0; i < nPI; i++){
    for(j = 0; j < pow(2,IdxPI[i][0]); j++){
        if(DecIdxCount[Column[IdxPI[i][0]][IdxPI[i][1]][IdxPI[i][2]][nVariable+3+
IdxPI[i][0]+j]] != 0){
            j = pow(2,IdxPI[i][0]);
            reduksiPIChartY[nSisaPI] = IdxPI[i];
            nSisaPI++;
        }
    }
}

//---reduksiPIChart[i][j] Menyimpan Nilai Yang Menandakan Bahwa PI
('1'berarti sudah diambil, '0' berarti belum diambil)---//
if(nSisaPI != 0){
    for(i = 0; i < nSisaMT; i++){
        reduksiPIChart[i] = (int *)malloc(nSisaPI * sizeof(int));
    }

    for(i = 0; i < nSisaMT; i++){
        for(j = 0; j < nSisaPI; j++){
            reduksiPIChart[i][j] = 0;
        }
    }

    for(i = 0; i < nSisaPI; i++){
        for(j = 0; j < pow(2, reduksiPIChartY[i][0]); j++){
            for(k = 0; k < nSisaMT; k++){
                if(Column[reduksiPIChartY[i][0]][reduksiPIChartY[i][1]][reduksiPIChartY[i]
][2]][nVariable+3+reduksiPIChartY[i][0]+j] == reduksiPIChartX[k]){
                    reduksiPIChart[k][i] = 1;
                }
            }
        }
    }
}

//---Memilih EPI Dari PI Chart yang Sudah Tereduksi---//
int *chooseEPI;

// Alokasi Memory Serta Inisialisasi Nilai Variabel chooseEPI

```

```

        chooseEPI = (int *)malloc(nSisaMT*sizeof(int)); // chooseEPI[i]
digunakan sebagai input untuk fungsi Recursion_For_Loop
        for(i = 0; i < nSisaMT; i++){
            chooseEPI[i] = -1;
        }

        int nPossibleEPI = 1;

        for(i = 0; i < nSisaMT; i++){
            nPossibleEPI = nPossibleEPI *
DecIdxCount[reduksiPIChartX[i]];
        }

        int **potentialEPI;

        // Alokasi Memory Serta Inisialisasi Nilai Variabel potentialEPI
        potentialEPI=(int **)malloc(nPossibleEPI * sizeof(int*));
        for(i = 0; i < nPossibleEPI; i++){
            potentialEPI[i] = (int *)malloc(nSisaMT*sizeof(int));
        }

        recursive(nSisaMT-1, nSisaMT, nSisaPI, reduksiPIChart, chooseEPI,
potentialEPI);

        int *NoOfPIForEPI;

        // Alokasi Memory Serta Inisialisasi Nilai Variabel NoOfPIForEPI
        NoOfPIForEPI = (int *)malloc(nPossibleEPI * sizeof(int)); //
NoOfPIForEPI[i] digunakan untuk menghitung banyaknya PI yang terdapat
pada setiap kombinasi yang mengcover seluruh minterm
        for(i = 0; i < nPossibleEPI; i++){
            NoOfPIForEPI[i] = 0;
        }

        for(i = 0; i < nPossibleEPI; i++){
            for(j = 0; j < nSisaMT; j++){
                if(potentialEPI[i][j] != -1){
                    NoOfPIForEPI[i]++;

                    for(k = j+1; k < nSisaMT; k++){
                        if(potentialEPI[i][k] == potentialEPI[i][j]){
                            potentialEPI[i][k] = -1;
                        }
                    }
                }
            }
        }

        //---Mencari Kombinasi yang Hanya Membutuhkan Minterm Paling
Sedikit dari PI untuk Mengcover Seluruh Minterm---//
        int minComb = 0;
        for(i = 1; i < nPossibleEPI; i++){
            if(NoOfPIForEPI[i] < NoOfPIForEPI[minComb]){
                minComb = i;
            }
        }

        for(i = 0; i < nSisaMT; i++){
            if(potentialEPI[minComb][i] != -1){
                IdxEPI[nEPI++] =
reduksiPIChartY[potentialEPI[minComb][i]];
            }
        }

```

```
        }  
    }  
}  
  
//---Menampilkan Hasil Akhir Minimisasi Logic dalam Bentuk SOP---//  
minResult(nVariable, nEPI, IdxEPI, Column);  
  
return 0;  
}
```


4. KESIMPULAN DAN LESSON LEARNED

Secara umum, permasalahan yang terdapat pada tugas besar Mata Kuliah Pemecahan Masalah Dengan C (EL2008) adalah mengenai minimisasi fungsi logika. Sehingga secara garis besar, kita diminta untuk membuat program yang dapat melakukan minimisasi suatu fungsi logika. Minimisasi fungsi logika merupakan suatu proses menyederhanakan suatu fungsi *boolean algebra*. Setiap fungsi *boolean algebra* dapat dinyatakan dalam bentuk *sum of product* (minterm) atau *product of sum* (maxterms). Dalam melakukan penyederhanaan fungsi ekspresi *boolean algebra* dapat menggunakan beberapa cara, diantaranya yaitu metode manipulasi *boolean algebra*, Karnaugh Maps, dan Quine-McCluskey atau Metode Tabular. Beberapa literatur merekomendasikan penggunaan metode Quine-McCluskey karena dianggap paling baik dalam penyederhanaan dan dapat digunakan untuk variable fungsi *boolean algebra* yang besar. Metode tabulasi menggunakan tahap-tahap penyederhanaan yang jelas dan teratur dalam penyederhanaan suatu fungsi aljabar.

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.

5. PEMBAGIAN TUGAS DALAM KELOMPOK

Berikut ini merupakan tabel pembagian tugas dari kelompok kami dalam pengerjaan tugas besar pada Mata Kuliah Pemecahan Masalah Dalam Bahasa C.

Anggota	Kontribusi				
	Pembuatan File Presentasi	Pembuatan File Laporan	Pembuatan Flowchart	Pembuatan Data Flow Diagram	Pembuatan Source Code
Farhan Hakim Iskandar (13220007)					
Fitra Nurindra (13220011)					
Muhammad Daffa Daniswara (13220043)					

DAFTAR REFERENSI

- Agung, Fajri Septia., dkk. 2009. *Sistem Deteksi Asap Rokok Pada Ruangan Bebas Asap Rokok Dengan Keluaran Suara*. <https://media.neliti.com>. Diakses pada 16 April 2022.
- Almanda, Deni, Krisdianto Krisdianto, and Erwin Dermawan. *Manajemen Konsumsi Energi Listrik Dengan Menggunakan Sensor PIR dan LM 35*.
- Chang, H.-Y., Wu, H.-K., & Hsu, Y.-S. (2013). *Integrating a mobile augmented reality activity to contextualize student learning of a socioscientific issue*. *British Journal of Educational Technology* , 44 (3), E95–E99.
- Desyantoro, Eka., Adian Fatchur Rochim, dan Kurniawan Teguh Martono. 2015. *Sistem Pengendali Peralatan Elektronik dalam Rumah Secara Otomatis Menggunakan Sensor PIR, Sensor LM35, dan Sensor LDR*. <https://media.neliti.com>. Diakses pada 16 April 2022.
- Fandi Rosi Sarwo Edi. 2016. *Teori Wawancara Psikopdiagnostik* (Yogyakarta: Leutikaprio). https://www.google.co.id/books/edition/Teori_Wawancara_Psikodignostik/uS96DwAAQBAJ?hl=id&gbpv=1. Diakses tanggal 21 April 2022.
- Harrison, R., Flood, D., & Duce, D. (2013). *Usability of mobile applications: literature review and rationale for a new usability model*. *Journal of Interaction Science*, 1(1), 1. (<https://doi.org/10.1186/2194-0827-1-1>)
- Hughes-Cromwick, E. L. (1985). *Nairobi Households and Their Energy Use: An Economic Analysis of Consumption Patterns, Energy Economics*, Vol 7, 265-278.
- Jadhav, J., Pratiksha, S., & Sunita, G. (2016). *Map Application Using Augmented Reality Technology for Smart Phones*. *International Journal of Science and Engineering Applications* , 5 (8), 421-424.
- Lee, V & Schneider, H & Schell, R. 2004. *Mobile Applications : Architecture, Design & Development*. Prentice Hall PTR : New Jersey.
- Nugroho, Agung. 2006. *Metode Pengaturan Penggunaan Tenaga Listrik dalam upaya Penghematan bahan bakar Pembangkit dan Energi*. <https://ejournal.undip.ac.id>. Diakses pada 16 April 2022.
- Pitts, A. & Ashby, R. (2011). *A Study of the Relationships between Income, Energy Consumption and Home Insulation Installation*. *UK Energy Research Centre Conference: Energy and People: Futures, Complexity and Challenge*, September 2011, Oxford.
- Putra, Agfianto E., 2012. *Mikrokontroler DSP & Embedded Electronics*. <http://agfi.staff.ugm.ac.id>. Diakses pada 16 April 2022.
- Setiandito, Yoga., dkk. 2011. *Thermal Sensor LM35*. Universitas Kristen Maranatha, Bandung. <https://media.neliti.com>. Diakses pada 16 April 2022.
- Santoso, Arif Dwi dan M. Agus Salim. *Penghematan Listrik Rumah Tangga dalam Menunjang Kestabilan Energi Nasional dan Kelestarian Lingkungan*. <http://ejurnal.bppt.go.id>. Diakses pada 16 April 2022.