

IMPLEMENTASI ALGORITME *BRUTE FORCE* DALAM BAHASA PEMROGRAMAN C UNTUK MENYELESAIKAN PERMAINAN KARTU **24**

Disusun untuk memenuhi laporan tugas mata kuliah IF2211
Strategi Algoritma semester 4 di Institut Teknologi Bandung.



Disusun oleh:

Ryan Samuel Chandra
13521140

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

PRAKATA

Puji dan syukur saya panjatkan ke hadirat Tuhan Yang Maha Esa, karena berkat-Nya begitu melimpah dalam kehidupan saya. Akhirnya, saya dapat menyelesaikan laporan ini tepat waktu setelah seminggu penuh melakukan pemrograman sekaligus riset dan pengujian. Saya juga mengucapkan terima kasih kepada pihak ITB, khususnya Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., serta seluruh Kakak-Kakak asisten mata kuliah IF2211 Strategi Algoritma, yang telah memberikan saya kesempatan untuk mengerjakan tugas ini.

Berikut disajikan laporan lengkap tentang “Implementasi Algoritme *Brute Force* dalam Bahasa Pemrograman C untuk Menyelesaikan Permainan Kartu 24”. Laporan ini selain dibuat agar dapat bermanfaat bagi masyarakat, juga secara khusus disusun untuk memenuhi salah satu tugas besar dalam mata kuliah IF2211 Strategi Algoritma di semester 4 Teknik Informatika Institut Teknologi Bandung.

Dengan berbagai sumber acuan, serta beberapa percobaan secara mandiri, saya berusaha sebaik mungkin untuk memenuhi tujuan pembuatan yang telah ditetapkan di tengah hambatan lingkungan dan tekanan waktu.

Saya berharap laporan ini dapat menjadi sesuatu yang memuaskan bagi semua pembaca. Tetapi saya menyadari bahwa laporan ini masih jauh dari kesempurnaan. Untuk itu, saya mohon kritik dan saran yang bersifat konstruktif untuk perkembangan percobaan saya.

Bandung, 24 Januari 2023

Penyusun Laporan

Daftar Isi

Prakata	ii
Daftar Isi	iii
BAB 1 : TINJAUAN PUSTAKA	1
1.1 Permainan Kartu 24	1
1.2 Algoritme <i>Brute Force</i>	1
1.3 Permutasi	2
BAB 2 : IMPLEMENTASI	3
2.1 Algoritme <i>Brute Force</i> dalam Menyelesaikan <i>24-Game</i>	3
2.2 <i>Abstract Data Type</i> : Matriks	3
2.3 Fungsi-Fungsi Pendukung	5
2.4 Program Utama	6
BAB 3 : DOKUMENTASI EKSPERIMEN	10
BAB 4 : PENUTUP	17
4.1 Kesimpulan	17
4.2 Saran	17
4.3 Refleksi	17
LAMPIRAN	18
DAFTAR REFERENSI	19

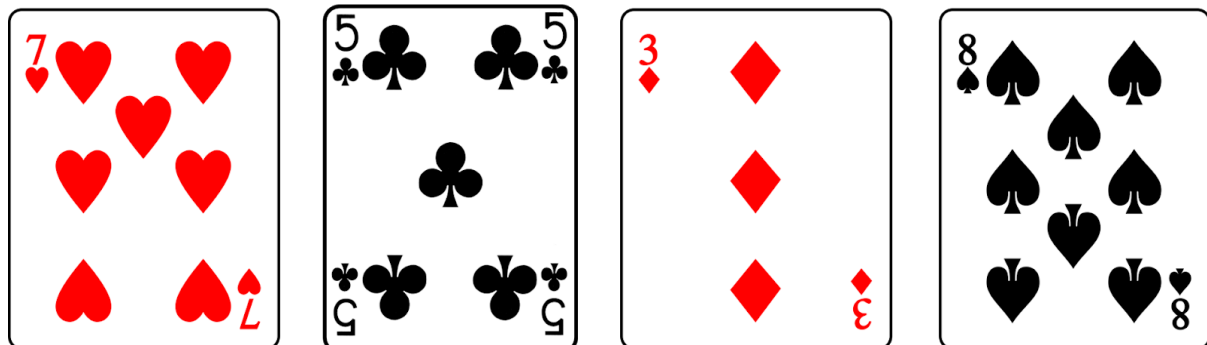
BAB 1

TINJAUAN PUSTAKA

1.1 Permainan Kartu 24

Permainan kartu 24 (*24-game*) adalah salah satu permainan aritmatika kartu remi sederhana yang cocok dimainkan pada saat berkumpul bersama teman, keluarga, atau kelompok. Permainan tersebut dapat mengasah otak dan melatih kecepatan berhitung seseorang. Dalam memainkannya, masing-masing pemain harus menemukan cara untuk menghitung empat bilangan bulat dari kartu yang dikeluarkan, sehingga hasil akhirnya adalah 24. Berikut adalah cara dan aturan bermain permainan kartu 24:

1. Pemain dikumpulkan, boleh dua atau lebih, seorang diri juga bisa.
2. Tumpukan kartu remi tanpa kartu *joker* diletakkan di tengah. Pemain yang merangkap sebagai “moderator” membuka empat kartu teratas dari tumpukan. Contoh, kartu yang terbuka adalah 7, 3, 5, dan 8.
3. Setiap pemain bertugas mengoperasikan empat kartu tersebut sehingga menghasilkan 24 dengan hanya menggunakan operator kali (\times), bagi ($/$), tambah ($+$), dan kurang ($-$).
4. Setiap kartu harus digunakan tepat satu kali. Sebagai contoh, keempat angka pada nomor 2 dapat menghasilkan 24 dengan cara: $7 \times 5 - (8 + 3)$.
5. Jika semua pemain merasa keempat kartu tersebut tidak dapat dikalkulasikan menjadi 24, empat kartu tersebut dikembalikan ke tumpukan, kemudian tumpukan diacak.



Gambar 1 Contoh kartu yang muncul dan harus dioperasikan menjadi 24
(Sumber: <https://hakonyoa.blogspot.com/2020/04/main-game-24-sederhana.html>)

Kartu remi sendiri terdiri dari 52 kartu yang terbagi menjadi empat suit (sekop, hati, keriting, dan wajik) yang masing-masing terdiri dari 13 kartu (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). Yang perlu diperhatikan hanyalah nilai kartu yang didapat (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). As bernilai 1, Jack bernilai 11, Queen bernilai 12, King bernilai 13, sedangkan kartu bilangan memiliki nilai dari bilangan itu sendiri ^[3].

1.2 Algoritme *Brute Force*

Algoritme *brute force* adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan ^[4]. Biasanya algoritme ini didasarkan pada pernyataan pada persoalan (*problem statement*) dan definisi/konsep yang dilibatkan. *Brute force* memecahkan persoalan dengan sangat sederhana, langsung, dan jelas caranya.

Algoritme *brute force* umumnya tidak “cerdas” dan tidak mangkus, karena ia membutuhkan volume komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Kata “*force*” mengindikasikan tenaga ketimbang otak. Maka dari itu, kadang-kadang algoritme ini disebut juga algoritme naif ^[4].

Algoritme *brute force* lebih cocok untuk persoalan yang masukannya kecil, serta sering digunakan sebagai basis pembandingan dengan algoritme lain yang lebih mangkus. Meskipun bukan metode *problem solving* yang mangkus, hampir semua persoalan dapat diselesaikan secara akurat dengan algoritme ini ^[4].

1.3 Permutasi

Permutasi adalah susunan unsur berbeda yang dibentuk dari n unsur, diambil dari n unsur atau sebagian unsur ^[5]. Permutasi dapat dikelompokkan menjadi beberapa macam. Di sini akan dibahas beberapa yang diperlukan dalam membuat program *24-game*.

1. Permutasi dari n elemen, tiap permutasi terdiri dari n elemen

Jika ada n unsur yang berbeda, diambil n unsur, maka banyaknya susunan (permutasi) yang berbeda dari n unsur tersebut adalah:

$${}_nP_n = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!$$

dengan $n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$.

2. Permutasi n elemen, tiap permutasi terdiri dari r unsur dari n elemen dengan $r \leq n$

Untuk semua bilangan positif n dan r , dengan $r \leq n$, banyaknya permutasi dari n objek yang diambil r objek pada satu waktu adalah:

$${}_nP_r = \frac{n!}{(n-r)!}$$

3. Permutasi berulang dari n unsur, tiap permutasi terdiri dari k unsur

$$P_n = n^k$$

BAB 2

IMPLEMENTASI

2.1 Algoritme *Brute Force* dalam Menyelesaikan *24-Game*

Seperti yang sudah dituliskan sebelumnya, algoritme *brute force* memiliki penyelesaian yang lempang, alias menguji semua kemungkinan sampai menemukan hasil yang diinginkan. Maka dari itu, seluruh solusi permainan kartu 24 dapat ditemukan secara *brute force*, dengan menaruh keempat kartu dalam susunan tertentu, lalu mencoba menerapkan setiap hasil permutasi berulang dari operator tambah, kurang, kali, dan bagi.

Sebagai contoh, kartu yang keluar adalah 7, 3, 5, 8, seperti contoh sebelumnya. Maka, dengan menggunakan rumus permutasi, kita mengetahui bahwa ada ${}_4P_4$ susunan yang mungkin

$${}_4P_4 = \frac{4!}{(4-4)!} = \frac{4 \times 3 \times 2 \times 1}{0!} = \frac{24}{1} = 24 \text{ susunan}$$

yaitu [7, 3, 5, 8], [7, 3, 8, 5], [7, 5, 3, 8], [7, 5, 8, 3], [7, 8, 3, 5], [7, 8, 5, 3], [3, 7, 5, 8], dan seterusnya. Sedangkan jumlah susunan operator pun selalu sama, yaitu dengan menggunakan rumus permutasi berulang, didapatkan $4^3 = 64$ susunan, yaitu [+ , + , +], [+ , + , -], [+ , + , ×], [+ , + , /], [+ , - , +], [+ , - , -], dan seterusnya.

Semua hasil di atas dapat disimpan didalam dua buah senarai yang berbeda, yang kemudian menjadi senarai dalam senarai (matriks). Selanjutnya, program hanya perlu diinstruksikan untuk menjalankan dua buah kalang yang mengiterasi semua elemen senarai satu per satu. Misalnya, susunan [7, 3, 5, 8] dioperasikan menggunakan [+ , + , +], menjadi “7 + 3 + 5 + 8 = 23”. Maka, formasi tersebut diabaikan karena hasilnya tidak 24. Kemudian, program mencoba “7 + 3 + 5 - 8 = 7”, dan tetap mengabaikannya. Proses tersebut terus diulangi sampai ujung kedua senarai.

Tidak lupa, ada tanda kurung yang dapat memengaruhi urutan pengoperasian. Dengan 4 angka, hanya ada 5 kemungkinan pengurangan. Sebagai contoh untuk [a, b, c, d] dan [+ , + , +], harus diperiksa:

- | | | |
|----------------------|----------------------|----------------------|
| 1. ((a + b) + c) + d | 3. a + ((b + c) + d) | 5. (a + b) + (c + d) |
| 2. (a + (b + c)) + d | 4. a + (b + (c + d)) | |

menghasilkan total $24 \times 64 \times 5 = 7680$ persamaan yang harus dievaluasi oleh komputer.

Apabila sebuah persamaan ternyata menghasilkan 24, maka persamaan tersebut akan ditambahkan ke senarai baru. Dalam bahasa C, *string* tidak bisa dioperasikan sehingga persamaan yang memenuhi terpaksa harus dimasukkan karakter demi karakter ke dalam sebuah matrix yang elemennya *character*.

Detail minor yang menjadi perhatian adalah jumlah solusi yang dimulai dari 0, lalu ditambah satu setiap ada persamaan yang memenuhi. Selain itu, digunakan pula *library* “time.h” untuk membantu perhitungan waktu eksekusi dari awal sampai akhir.

2.2 *Abstract Data Type*: Matriks

Menurut KBBI, istilah “matriks” dalam dunia komputasi berarti larik dua dimensi yang disusun dalam baris dan kolom, yang nilai baris dan kolomnya dapat sama atau berbeda. Maka dari itu, setiap elemen matriks terdefinisi berdasarkan dua buah indeks yang secara berurutan

mengacu tepat pada sebuah baris dan sebuah kolom pada matriks. Indeks harus bertipe mempunyai suksesor dan predesesor, misalnya *integer*.

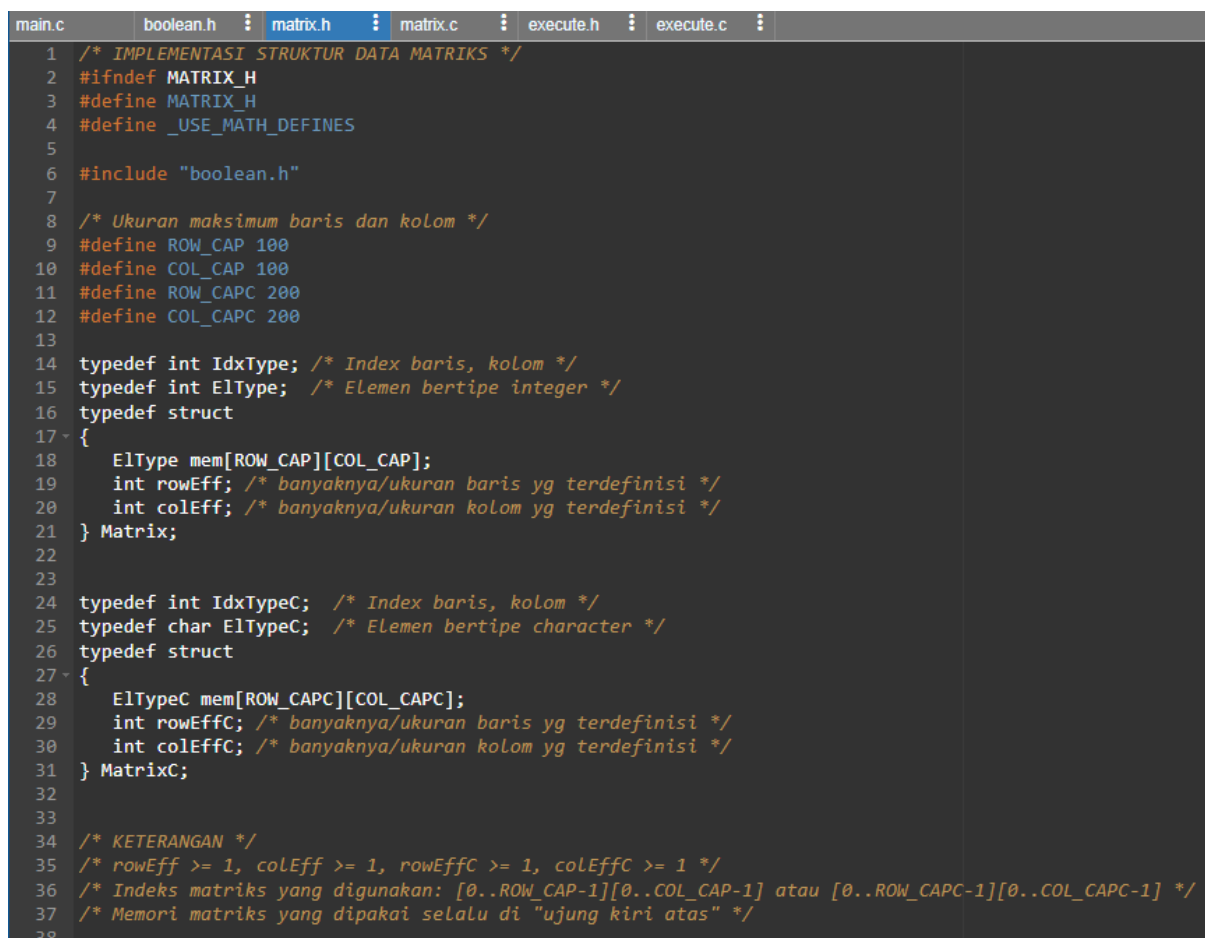
Matriks adalah struktur data statis, sehingga ukuran memori yang disediakan harus ditentukan di awal. Di sini, ukuran maksimalnya adalah 100×100 untuk matriks yang berelemen *integer*, dan 200×200 untuk matriks berelemen *character*. Ukuran setiap matriks dalam program tidak langsung mengikuti maksimalnya, melainkan akan dideklarasikan sesuai dengan kebutuhan.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1j} \\ a_{21} & a_{22} & \dots & a_{2j} \\ \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \dots & a_{ij} \end{bmatrix}$$

Gambar 2 Contoh matriks dengan a_{ij} sebagai elemen terurut

(Sumber: <https://stkiptsb.ac.id/simak-tsb/admin/pubek/15355669045.%20BUKU%20aljabar%20linear%202016.pdf>)

Dalam program kali ini, fungsi utama dari matriks adalah menampung hasil permutasi kartu dan operator dalam bentuk angka (representatif). Selain itu, persamaan-persamaan yang menghasilkan 24 pun akan disimpan dalam sebuah matriks yang berbeda, untuk kemudian ditampilkan bersamaan ke layar setelah semua perhitungan selesai.



```

main.c  boolean.h  matrix.h  matrix.c  execute.h  execute.c
1  /* IMPLEMENTASI STRUKTUR DATA MATRIKS */
2  #ifndef MATRIX_H
3  #define MATRIX_H
4  #define _USE_MATH_DEFINES
5
6  #include "boolean.h"
7
8  /* Ukuran maksimum baris dan kolom */
9  #define ROW_CAP 100
10 #define COL_CAP 100
11 #define ROW_CAPC 200
12 #define COL_CAPC 200
13
14 typedef int IdxType; /* Index baris, kolom */
15 typedef int ElType; /* Elemen bertipe integer */
16 typedef struct
17 {
18     ElType mem[ROW_CAP][COL_CAP];
19     int rowEff; /* banyaknya/ukuran baris yg terdefinisi */
20     int colEff; /* banyaknya/ukuran kolom yg terdefinisi */
21 } Matrix;
22
23
24 typedef int IdxTypeC; /* Index baris, kolom */
25 typedef char ElTypeC; /* Elemen bertipe character */
26 typedef struct
27 {
28     ElTypeC mem[ROW_CAPC][COL_CAPC];
29     int rowEffC; /* banyaknya/ukuran baris yg terdefinisi */
30     int colEffC; /* banyaknya/ukuran kolom yg terdefinisi */
31 } MatrixC;
32
33
34 /* KETERANGAN */
35 /* rowEff >= 1, colEff >= 1, rowEffC >= 1, colEffC >= 1 */
36 /* Indeks matriks yang digunakan: [0..ROW_CAP-1][0..COL_CAP-1] atau [0..ROW_CAPC-1][0..COL_CAPC-1] */
37 /* Memori matriks yang dipakai selalu di "ujung kiri atas" */
38

```

Gambar 3 Tipe data bentukan: “Matrix” dan “MatrixC” dalam “matrix.h”

(Sumber: dokumentasi penulis)


```

41 /* *** Konstruktor membentuk Matrix *** */
42 void createMatrix(int nRows, int nCols, Matrix *m);
43 void createMatrixC(int nRows, int nCols, MatrixC *m);
44 /* Membentuk sebuah Matrix "kosong" yang siap diisi berukuran nRow x nCol di "ujung kiri" memori */
45 /* I.S. nRow dan nCol adalah valid untuk memori matriks yang dibuat */
46 /* F.S. Matriks m sesuai dengan definisi di atas terbentuk */
47
48 /* *** Selektor *** */
49 #define ROW_EFF(M) (M).rowEff
50 #define COL_EFF(M) (M).colEff
51 #define ELMT(M, i, j) (M).mem[(i)][(j)]
52 #define ROW_EFFC(M) (M).rowEffC
53 #define COL_EFFC(M) (M).colEffC
54 #define ELMTC(M, i, j) (M).mem[(i)][(j)]
55
56 void displayMatrix(Matrix m);
57 void displayMatrixC(MatrixC m);
58 /* I.S. m terdefinisi */
59 /* F.S. Nilai m(i,j) ditulis ke layar per baris per kolom */
60
61 #endif

```

Gambar 4 Fungsi dan selektor ADT matriks dalam “matrix.h”

(Sumber: dokumentasi penulis)

Implementasi lengkap dari masing-masing fungsi (prosedur) bisa dilihat dalam file “matrix.c”, yang terdapat pada tautan yang dicantumkan di bagian **Lampiran**.

2.3 Fungsi-Fungsi Pendukung

Dalam membuat sebuah program, modularitas sangatlah penting agar program lebih efisien, mudah dibaca, dan tidak menumpuk di satu tempat. Maka dari itu, program *24-game solver* ini dibagi ke dalam fungsi-fungsi kecil yang memiliki peranan masing-masing. Fungsi-fungsi tersebut dapat dipanggil berkali-kali saat dibutuhkan.

```

main.c | boolean.h | matrix.h | matrix.c | execute.h | execute.c
1 #include "boolean.h"
2
3 /* PERNYATAAN SUMBER */
4 /* swap & permutation mengacu pada: https://people.engr.tamu.edu/djimenez/ut/utsa/cs3343/Lecture25.html */
5 /* makeOpsCombo mengacu pada : https://rosettacode.org/wiki/Permutations_with_repetitions#C */
6
7 /* Kelompok fungsi mencari tempat kosong */
8 int searchAvailable (Matrix m);
9 int searchAvailableC (MatrixC m);
10 int arrSearchAvailable (int a[], int n);
11
12 /* Kelompok fungsi menerima dan membuat input */
13 boolean validateInput(char w[]);
14 void makeInputs (char arr[], int out[]);
15 void make_from_random (int num, int out[]);
16
17 /* Kelompok fungsi permutasi */
18 void swap (int arr[], int i, int j);
19 void permutation (int v[], int n, int i, Matrix *result);
20 boolean isDuplicate (Matrix m, int x);
21 void makeOpsCombo (Matrix *result);
22
23 /* Kelompok fungsi operasi */
24 float operation (float a, float b, int op);
25 void setElmtC (MatrixC *valid_combo, Matrix num_combo, int num, int no, int row, int *col);
26
27 /* Kelompok fungsi evaluasi */
28 void evaluate1 (Matrix num_combo, Matrix op_combo, MatrixC *valid_combo, int num, int op, int *num_of_solution);
29 void evaluate2 (Matrix num_combo, Matrix op_combo, MatrixC *valid_combo, int num, int op, int *num_of_solution);
30 void evaluate3 (Matrix num_combo, Matrix op_combo, MatrixC *valid_combo, int num, int op, int *num_of_solution);
31 void evaluate4 (Matrix num_combo, Matrix op_combo, MatrixC *valid_combo, int num, int op, int *num_of_solution);
32 void evaluate5 (Matrix num_combo, Matrix op_combo, MatrixC *valid_combo, int num, int op, int *num_of_solution);
33
34 /* Kelompok fungsi mencetak */
35 void print_from_num (int num);
36 char num_to_op (int num);
37 void print_from_num2 (int num, FILE *file);
38 void printToFile (int inputs[], MatrixC m, int num_of_solution, double execution_time);

```

Gambar 5 Deklarasi fungsi-fungsi pendukung dalam “execute.h”
(Sumber: dokumentasi penulis)

Berikut adalah deskripsi singkat dari masing-masing fungsi:

1. **searchAvailable** : mengembalikan indeks terkecil dari baris yang siap diisi pada sebuah matriks berelemen *integer*. Siap diisi berarti semua elemennya bernilai 0.
2. **searchAvailableC** : mengembalikan indeks terkecil dari baris yang siap diisi pada sebuah matriks berelemen *character*. Siap diisi berarti semua elemennya kosong.
3. **arrSearchAvailable** : mengembalikan indeks terkecil dari “tempat” yang siap diisi pada sebuah senarai berelemen *integer*. Siap diisi berarti elemennya bernilai 0.
4. **validateInput** : memvalidasi keempat masukan dari pengguna. Input yang valid hanyalah “A”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “10”, “J”, “Q”, “K”.
5. **makeInputs** : merepresentasikan masing-masing masukan pengguna sebagai sebuah *integer* dan memasukannya dalam sebuah senarai.
6. **make_from_random** : membuat senarai “masukan” dari 4 angka (dari 1 sampai dengan 13) yang dibangkitkan secara acak oleh komputer.
7. **swap** : pendukung fungsi permutasi. Menukar dua buah elemen dalam senarai *integer*.
8. **permutation** : memasukkan semua kemungkinan susunan kartu (dalam representasi bilangan bulat) ke dalam sebuah matriks (per baris).
9. **isDuplicate** : memeriksa apakah sebuah baris yang ditunjuk dalam sebuah matriks memiliki duplikat dalam matriks itu sendiri.
10. **makeOpsCombo** : memasukkan semua kemungkinan susunan operator (dalam representasi bilangan bulat) ke dalam sebuah matriks (per baris).
11. **operation** : menerima parameter berupa tiga angka, kemudian mengoperasikan dua angka pertama dengan operator yang kodenya sesuai dengan angka ketiga.
12. **setElmtC** : mengubah kembali sebuah angka menjadi bentuk kartunya (misalnya “1” menjadi “A”), lalu menambahkannya ke dalam matriks keluaran pada kolom kosong di baris tertentu, kemudian menggeser indeks kolom kosongnya.
13. **evaluate1** : evaluasi pola persamaan $((a_b)_c)_d$, lalu menambahkannya ke senarai keluaran apabila hasilnya sama dengan 24.
14. **evaluate2** : evaluasi pola persamaan $a_(b_c))_d$, lalu menambahkannya ke senarai keluaran apabila hasilnya sama dengan 24.
15. **evaluate3** : evaluasi pola persamaan $a_((b_c)_d)$, lalu menambahkannya ke senarai keluaran apabila hasilnya sama dengan 24.
16. **evaluate4** : evaluasi pola persamaan $a_(b_ (c_d))$, lalu menambahkannya ke senarai keluaran apabila hasilnya sama dengan 24.
17. **evaluate5** : evaluasi pola persamaan $(a_b)_(c_d)$, lalu menambahkannya ke senarai keluaran apabila hasilnya sama dengan 24.
18. **print_from_num** : mencetak ke layar, bentuk kartu dari angka tertentu. Misalnya, angka “13” akan mencetak “K”, sedangkan “1” akan mencetak “A”.
19. **num_to_op** : menerima parameter berupa sebuah *integer* dan mengembalikan *character* berupa operator yang representatifnya adalah *integer* tersebut.
20. **print_from_num2** : mencetak ke *file*, bentuk kartu dari angka tertentu. Misalnya, angka “13” akan mencetak “K”, sedangkan “1” akan mencetak “A”.
21. **printToFile** : prosedur yang langsung mencetak segala solusi dan perhitungan ke dalam sebuah *file* dengan nama yang diminta dari pengguna.


```

54  /* Meminta atau mengacak masukan */
55  printf("Apakah Anda ingin memasukkan angka sendiri? (Ketik '0'/'1'): ");
56  scanf("%d", &question);
57  while (go == 0) {
58      if (question == 1) {
59          while (valid_input == 0) {
60              printf("\nMasukkan kartu ke-1: ");
61              scanf("%s", w);
62              printf("Masukkan kartu ke-2: ");
63              scanf("%s", x);
64              printf("Masukkan kartu ke-3: ");
65              scanf("%s", y);
66              printf("Masukkan kartu ke-4: ");
67              scanf("%s", z);
68
69              if (validateInput(w) && validateInput(x) && validateInput(y) && validateInput(z)) {
70                  valid_input = 1;
71                  makeInputs(w, inputs);
72                  makeInputs(x, inputs);
73                  makeInputs(y, inputs);
74                  makeInputs(z, inputs);
75                  go = 1;
76              } else {
77                  printf("\nMasukan Anda tidak valid!");
78                  printf("\nMasukan yang valid: A,2,3,4,5,6,7,8,9,10,J,Q,K\n\n");
79              }
80          }
81      } else if (question == 0) {
82          lower = 1;
83          upper = 13;
84          srand (time(NULL));
85          random1 = (rand() % (upper - lower + 1)) + lower;
86          random2 = (rand() % (upper - lower + 1)) + lower;
87          random3 = (rand() % (upper - lower + 1)) + lower;
88          random4 = (rand() % (upper - lower + 1)) + lower;
89          printf("\nKartu ke-1: ");
90          print_from_num(random1);
91          printf("\nKartu ke-2: ");
92          print_from_num(random2);
93          printf("\nKartu ke-3: ");
94          print_from_num(random3);
95          printf("\nKartu ke-4: ");
96          print_from_num(random4);
97          printf("\n");
98
99          make_from_random (random1, inputs);
100         make_from_random (random2, inputs);
101         make_from_random (random3, inputs);
102         make_from_random (random4, inputs);
103         go = 1;
104     } else {
105         printf("Masukan Anda tidak valid. Masukkan hanya '0' atau '1'!\n");
106         printf("Apakah Anda ingin memasukkan angka sendiri? (Ketik '0'/'1'): ");
107         scanf("%d", &question);
108     }
109 }

```

```

112     /* Memulai pencarian solusi */
113     start = clock();
114     permutation (inputs, 4, 0, &num_combo);
115     for (int i = 0; i < 24; i++) {
116         if (isDuplicate(num_combo, i)) {
117             for (int j = 0; j < 4; j++) {
118                 ELMT(num_combo, i, j) = 0;
119             }
120         }
121     }
122     makeOpsCombo(&op_combo);
123
124     for (num = 0; num < 24; num++) {
125         for (op = 0; op < 64; op++) {
126             if (ELMT(num_combo, num, 0) != 0) {
127                 evaluate1 (num_combo, op_combo, &valid_combo, num, op, &num_of_solution);
128                 evaluate2 (num_combo, op_combo, &valid_combo, num, op, &num_of_solution);
129                 evaluate3 (num_combo, op_combo, &valid_combo, num, op, &num_of_solution);
130                 evaluate4 (num_combo, op_combo, &valid_combo, num, op, &num_of_solution);
131                 evaluate5 (num_combo, op_combo, &valid_combo, num, op, &num_of_solution);
132             }
133         }
134     }
135
136     end = clock();
137     execution_time = ((double)(end - start))/CLOCKS_PER_SEC;
138
139
140     /* Menampilkan solusi */
141     printf("\n");
142     printf("\n%d solusi ditemukan!\n", num_of_solution);
143     displayMatrixC(valid_combo);
144     printf("\nWaktu eksekusi: %.5f detik", execution_time);
145     printf("\n");
146
147     /* Mengurus fail */
148     go = 0;
149     while (go == 0) {
150         printf("\nApakah Anda ingin menyimpan solusi ke dalam fail? (Ketik '0'/'1'): ");
151         scanf("%d", &file_question);
152         if (file_question == 1) {
153             printToFile (inputs, valid_combo, num_of_solution, execution_time);
154             printf("\nTerima kasih sudah menggunakan layanan 24-GAME SOLVER :)");
155             go = 1;
156         } else if (file_question == 0) {
157             printf("Terima kasih sudah menggunakan layanan 24-GAME SOLVER :)");
158             go = 1;
159         } else {
160             printf("Masukan Anda tidak valid! Masukkan hanya '0' atau '1'!\n");
161         }
162     }
163
164     return (0);
165 }

```

Gambar 6 *Source code* program utama bagian per bagian
(Sumber: dokumentasi penulis)

BAB 4

PENUTUP

4.1 Kesimpulan

Permainan kartu 24 (*24-game*) adalah salah satu permainan aritmatika kartu remi sederhana dapat mengasah otak dan melatih kecepatan berhitung seseorang. Peraturannya pun amat mudah; setiap pemain cukup berlomba mengoperasikan 4 kartu teratas dari tumpukan sehingga dapat menghasilkan 24, hanya dengan operator kali (\times), bagi ($/$), tambah ($+$), kurang ($-$), serta tanda kurung ($()$) sebagai tambahan.

Solusi dari permainan tersebut dapat ditemukan dengan program komputer. Meskipun dalam praktiknya, para pemain tentu akan dengan cepat memikirkan cara yang paling mungkin untuk menyusun keempat kartu sedemikian rupa, namun komputer berbeda. Ia akan mencoba satu per satu susunan yang mungkin dari kartu dan operator. Algoritme seperti itu dinamakan *brute force*, yaitu pendekatan yang lempang (*straightforward*), jelas, langsung, dan sederhana untuk memecahkan suatu persoalan.

Dengan bantuan konstruksi tipe data matriks, beberapa fungsi pendukung, serta konsep permutasi, program *24-game solver* berhasil dibuat dalam bahasa pemrograman C. Program lengkap dapat dilihat pada tautan di bagian **LAMPIRAN**, sedangkan beberapa contoh eksekusi ada di bagian **BAB 3**.

4.2 Saran

1. Sebaiknya, program seperti *24-game solver* dibuat dalam bahasa pemrograman yang lebih “ramah”/mudah dalam memproses *string*; tidak harus dibuat menjadi *array of character* seperti di C. Hal tersebut akan memudahkan dalam mengatur keluaran.
2. Dalam mengerjakan pemrograman, sebaiknya pemrogram memiliki catatan kecil tentang fungsi, prosedur, dan juga fitur yang telah dibuat. Hal ini bertujuan untuk mempermudah modifikasi program, serta agar tidak ada duplikat fungsi.
3. Membaca spesifikasi dengan teliti sampai tuntas sebaiknya dilakukan sebelum memulai pemrograman. Dengan demikian, tidak perlu ada perubahan massal di tengah pekerjaan karena ketidaksesuaian dengan spesifikasi.
4. Seharusnya, waktu yang diberikan untuk mengerjakan tugas ini lebih dari satu minggu, mengingat hari libur Imlek tidak dapat digunakan untuk bekerja karena sudah terisi.

4.3 Refleksi

Tugas besar ini memberikan saya banyak sekali pengalaman dan ilmu baru dalam pemrograman menggunakan C. Di sini pula saya belajar, bahwa pengaturan waktu dan koordinasi yang baik sangat diperlukan bahkan saat tidak bekerja dalam tim. Apabila satu fungsi saja tidak diimplementasikan dengan benar, maka keseluruhan program menjadi salah.

Dengan keterbatasan pengetahuan dan keahlian, saya pun jadi harus berusaha sangat keras mengeksplor lebih dan lebih dalam lagi mengenai tugas ini, baik dari internet, teman, juga saudara. Akhirnya, usaha saya tidak sia-sia. Tugas pemrograman pertama saya di semester 4 terselesaikan dengan baik dan tepat waktu.

LAMPIRAN

https://github.com/RyanSC06/Tucil1_13521140.git

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	
2. Program berhasil <i>running</i>	<input checked="" type="checkbox"/>	
3. Program dapat membaca input / generate sendiri dan memberikan luaran	<input checked="" type="checkbox"/>	
4. Solusi yang diberikan program memenuhi (berhasil mencapai 24)	<input checked="" type="checkbox"/>	
5. Program dapat menyimpan solusi dalam file teks	<input checked="" type="checkbox"/>	

DAFTAR REFERENSI

1. Rald. 3 April 2020 15.58. *MAIN GAME 24 SEDERHANA*. (Diakses tanggal 24 Januari 2023 dari <https://hakonyoa.blogspot.com/2020/04/main-game-24-sederhana.html>)
2. Bagoes, Putu. 1 Juni 2021 17.00. *4 Jenis Permainan Kartu yang Paling Seru dan Bisa Dimainkan Bersama Orang Terdekat*. (Diakses tanggal 24 Januari 2023 dari <https://kids.grid.id/read/472718891/4-jenis-permainan-kartu-yang-paling-seru-dan-bisa-dimainkan-bersama-orang-terdekat?page=all>)
3. Chandra, Evita. Bandung, 8 Desember 2015. *Penerapan Algoritma Brute Force pada Permainan Kartu 24 (24 game)*. (Diakses tanggal 24 Januari 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2015-2016/Makalah-2016/MakalahStima-2016-038.pdf>)
4. Munir, Rinaldi. 2022. *Algoritma Brute Force (Bagian 1)*. (Diakses tanggal 24 Januari 2023 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-\(2022\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf))
5. Heryansyah, T. Rizkha. 30 Agustus 2022. *Yuk, Belajar 5 Jenis Permutasi dalam Teori Peluang / Matematika Kelas 12*. (Diakses tanggal 25 Januari 2023 dari <https://www.ruangguru.com/blog/jenis-permutasi-dalam-teori-peluang>)