LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA
PENYELESAIAN PERMAINAN KARTU 24 DENGAN ALGORITMA *BRUTE FORCE*

Disusun Oleh

Michael Leon Putra Widhi (13521108)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

BAB I
ALGORITMA *BRUTE FORCE*

A. Algoritma *Brute Force* dalam pencarian solusi permainan kartu 24

Algoritma *Brute Force* adalah sebuah pendekatan yang mudah untuk memecahkan suatu masalah, biasanya didasarkan langsung pada pernyataan masalah dan definisi konsep yang dilibatkan atau lempang (*straight-forward*). Algoritma *Brute Force* memecahkan masalah dengan sangat sederhana, langsung, dan dengan cara yang cukup jelas. Algoritma ini menjadi dasar dari pembuatan program pencarian solusi penyelesaian permainan kartu 24. Permainan ini mengharuskan pemain untuk melakukan pencarian kemungkinan operasi yang dapat dinyatakan pada 4 buah kartu yang merepresentasikan sebuah nilai tertentu untuk mendapatkan nilai 24. Dengan pendekatan *Brute Force*, kita dapat melakukan pencocokan operasi dan bilangan secara satu per-satu hingga diperoleh semua solusi yang memenuhi kondisi tersebut.

B. Penjelasan Algoritma Program

Alur jalannya program adalah sebagai berikut. Pada saat pengguna melakukan *run* pada program, pengguna akan dihadapkan pada tampilan awal program. Pada bagian ini, pengguna akan diminta untuk memilih skema pemberian masukan kepada program, langsung dari pengguna atau melakukan pembangkitan 4 buah kartu secara acak. Program juga sudah dilengkapi dengan skema validasi masukan pada menu ini agar masukan sesuai dan mengarah hanya diantara dua pilihan tersebut.

Skema penerimaan masukan adalah sebagai berikut :

a. Masukan secara langsung dari pengguna

   Jika pengguna memilih menu ini, maka pengguna akan diminta untuk memberikan masukan sebanyak 4 buah kartu secara bebas. Program sudah dilengkapi dengan skema validasi masukan sehingga jika masukan pengguna salah, misalnya memasukkan karakter atau angka yang tidak terdefinisi pada permainan kartu 24 ($A,\ 2\ -\ 10,\ J,\ Q\ ,\ K$) atau memberikan masukan lebih dari 4 kartu, program akan memberikan peringatan, penjelasan secara singkat letak

kesalahan, dan kembali meminta masukan. Skema validasi ini terus dilakukan hingga masukan sudah benar dan diterima oleh program.

b. Melakukan pembangkitan 4 buah kartu secara acak

   Jika pengguna memilih menu ini, maka program akan secara otomatis melakukan pengacakan angka dan karakter yang mungkin dalam permainan kartu 24. Skema pembangkitan kartu acak ini dilakukan hingga keempat kartu terdefinisi dan benar. Pengguna tidak perlu memberikan masukan dan menunggu lama untuk skema validasi pembangkitan kartu ini karena dilakukan secara otomatis. Proses pembangkitan kartu ini dilakukan dengan melakukan pengacakan nilai bilangan pada rentang 1 hingga 13 yang merepresentasikan setiap nilai yang mungkin dalam kartu 24 dengan bantuan fungsi *Pseudo-random Number Generator* (PRNG) bentukan milik bahasa pemrograman C++ dan umpan berupa waktu saat program di *run*. Hal ini dilakukan demi memperbesar periode perulangan pembangkitan bilangan acak semu.

Setelah pengguna menjalankan skema penerimaan masukan, maka program akan melakukan pemrosesan terhadap masukan tersebut dengan skema *Brute Force* yang telah dirancang dan dibuat.

Adapun skema *Brute Force* yang digunakan adalah sebagai berikut :

1. [Inisialisasi] Melakukan pengecekan kombinasi yang mungkin dari keempat kartu yang menjadi masukan dengan prosedur *checkCombination*. Setiap kombinasi yang valid dan unik akan disimpan dalam 4 buah senarai yang menyimpan nilai masing-masing masukan beserta kombinasinya secara independen. Adapun jumlah kombinasi yang mungkin untuk 4 kartu unik adalah $4! = 24$ kombinasi.
2. Algoritma dimulai dengan pengaksesan elemen dalam senarai. Pada bagian ini, kalang *for* digunakan untuk mengakses elemen-elemen kombinasi masukan yang telah disimpan dalam 4 buah senarai pada langkah sebelumnya. Pengaksesan ini dilakukan guna diteruskan pada proses percobaan operator yang dilakukan menggunakan prosedur *bruteForce*.
3. Di dalam prosedur *bruteForce* dilakukan percobaan dengan menggunakan operator yang mungkin tanpa tanda kurung ($+$, $-$, $*$, $/$) untuk mengevaluasi

keempat nilai yang direpresentasikan kartu masukan secara satu per-satu hingga didapatkan nilai 24. Proses percobaan operator dilakukan dengan 3 buah kalang *for* bersarang dengan rentang nilai pada setiap kalang merepresentasikan operasi yang dilakukan oleh setiap operator pada nilai masukan. Adapun proses evaluasi nilai dilakukan dengan prosedur *processRes* yang berada pada ujung dari setiap perulangan pada kalang.

4. Pada prosedur *processRes* dilakukan proses evaluasi nilai dari setiap kombinasi masukan bilangan yang diberikan melalui masukan dan setiap kombinasi operator yang mungkin dari prosedur *bruteForce*. Pada bagian ini juga ditangani semua kombinasi tanda kurung yang mungkin dan valid dengan skema evaluasi dua pasang kurung. Misalkan a, b, c dan d adalah nilai masukan, dan op adalah sebuah operator, maka operasi kurung yang ditangani adalah :
    a. $((a\ op\ b)\ op\ c)\ op\ d$
    b. $(a\ op\ (b\ op\ c))\ op\ d$
    c. $a\ op\ ((b\ op\ c)\ op\ d)$
    d. $a\ op\ (b\ op\ (c\ op\ d))$
    e. $(a\ op\ b)\ op\ (c\ op\ d)$

    Selain melakukan operasi dan penanganan kurung, prosedur ini juga melakukan penyusunan keluaran dengan skema operator yang telah dicobakan dan konversi ke dalam bentuk operator dengan prosedur *operations* (sebelumnya didefinisikan sebagai nilai). Setiap kemungkinan kombinasi yang mungkin untuk menyusun nilai 24 kemudian dinyatakan sebagai solusi dan dimasukkan ke dalam senarai solusi.

Pada skema *Brute Force*, langkah 2 hingga 4 dilakukan secara berulang hingga seluruh kombinasi bilangan dalam senarai dan kombinasi operator telah dicobakan. Selama proses ini berlangsung, program melakukan kalkulasi waktu eksekusi dengan menggunakan fungsi bentukan dari modul *chrono* dalam bahasa pemrograman C++. Skema ini akan menghasilkan semua solusi yang memenuhi solusi permainan, tetapi belum dikelola secara mangkus karena belum menangani kasus duplikasi solusi.

Selanjutnya dilakukan skema unifikasi solusi dari solusi yang telah terdefinisi. Unifikasi dilakukan dengan melakukan inisialisasi senarai solusi unik global, kemudian

dilakukan pengecekan duplikasi solusi dengan menggunakan kalang *for* bersarang saling-berkait untuk kesangkilan pencarian. Elemen yang terduplikasi tidak akan dimasukkan ke dalam senarai solusi efektif. Setelah seluruh elemen senarai solusi selesai dianalisis, maka proses eksekusi telah selesai dilaksanakan dan program akan menampilkan hasil analisis sederhana berupa jumlah solusi dan waktu eksekusi program.

Setelah proses eksekusi selesai dilaksanakan, program akan meminta pengguna untuk memilih skema pengembalian luaran, langsung pada terminal atau disimpan dalam sebuah *file*. Program sudah dilengkapi dengan skema validasi masukan pada menu ini agar masukan sesuai dan mengarah hanya diantara dua pilihan tersebut. Hasil yang ditampilkan di terminal tidak akan disimpan dalam *file* dan berlaku sebaliknya. Solusi yang dikembalikan dalam bentuk *file* akan berada pada *folder test*.

BAB II

SOURCE CODE PROGRAM

A. *File header* (solver. hpp)

Pada *file* ini didefinisikan semua fungsi dan prosedur yang akan digunakan pada program utama. Berikut adalah tampilannya :

```
/* Header File solver.hpp */
#include <iostream>
#include <string>
using namespace std;

void randomNumber(int* nums);
// Membangkitkan angka secara anak (random number generator)
// Seeds yang digunakan adalah waktu saat ini, sehingga relatif sulit terulang kembali

void generate4(int* card1, int* card2, int* card3, int* card4);
// Mengenerate 4 angka random
// Menggunakan prosedur randomNumber yang telah diinisiasi sebelumnya

bool checkNumber(string str);
// Melakukan pengecekan apakah string yang masuk benar merupakan string atau integer
// Untuk memudahkan perlakuan konversi string ke integer

bool inputNumberValid (string input);
// Melakukan konversi dan validasi input berupa string ke integer
// Apakah valid (bernilai antara 2 - 10)

bool inputCharValid (string input);
// Melakukan analisis elemen pertama dari sebuah string
// Apakah valid (bernilai A, K, Q, atau J)

int chartoInt (string input);
// Melakukan konversi dari karakter karater yang mungkin dalam sebuah kartu menjadi sebuah integer

void inttoChar (int input, string *target);
// Melakukan konversi dari integer yang mungkin dalam sebuah kartu menjadi karakter kartu yang bersesuaian
```

Gambar 2.1.1. Beberapa fungsi dan prosedur pada *file header* solver.hpp – bagian (1).

```
char operations(int ops);
// Sebelumnya membuat representasi dari operasi dalam bentuk angka
// Fungsi ini mengembalikan bentuk angka dalam operasi menjadi operasi + - * /

float calculate (int ops, float x, float y);
// Melakukan operasi bilangan real antara dua bilangan yang representasi nya diubah
// menjadi bilangan real (untuk memudahkan real division) dan operator dalam angka

void checkCombination (int a, int b, int c, int d);
// Melakukan pengecekan terhadap semua kombinasi yang mungkin dari masukan a, b, c, d
// Kombinasi yang sudah ada tidak dimasukkan ke dalam senarai
// Adapun kombinasi yang belum ada akan dimasukkan ke dalam senarai analisis
```

Gambar 2.1.2. Beberapa fungsi dan prosedur pada *file header* solver.hpp – bagian (2).

```
void processRes (int p, int q, int r, int w, int x, int y, int z);
// Melakukan hasil kalkulasi dengan bilangan masukan w, x, y, z dan operator p, q, r
// Operator p, q, r dihasilkan melalui skema brute force operator yang mungkin untuk mencapai nilai 24

void bruteForce (int w, int x, int y, int z);
// Melakukan brute-force untuk menemukan pasangan solusi operator yang mungkin bagi bilangan
// w, x, y, dan z untuk mendapatkan nilai 24.
// Digunakan operator p, q, dan r dengan 3 kalang bersarang

void input ();
// Menangani proses input - masukan dari pengguna
// Pengguna dapat memilih untuk memasukkan input atau membangkitkan bilangan secara acak

void output ();
// Menangani proses output - kembalian ke pengguna
// Pengguna dapat memilih untuk hanya menampilkan di terminak atau menyimpan ke file
```

Gambar 2.1.3. Beberapa fungsi dan prosedur pada *file* header solver.hpp – bagian (3).

B. *File* program utama (solver. cpp)

Pada program utama diimplementasikan berbagai fungsi dan prosedur yang telah didefinisikan pada *file header.* Berikut adalah tampilannya :

1. Impor modul eksternal

```
/* File utama solver.cpp */
// Impor Modul Eksternal
#include <iostream>
#include "solver.hpp"
#include <cstdlib>
#include <chrono>
#include <fstream>
using namespace std;
using namespace std::chrono;
```

Gambar 2.2.1. Modul eksternal yang diimpor untuk membantu jalannya program.

2. Inisialisasi variabel global

```
// Inisialisasi Variabel Global
int a, b, c, d;        // Bilangan yang menjadi masukan untuk pemrosesan
string p, q, r, s;     // Inisialisasi dengan string 1 char untuk mempermudah konversi ke bilangan
// Inisialisasi senarai dengan 25 elemen yang mencakup semua kombinasi, 4! = 24 dengan 1 elemen block
int elem1[25], elem2[25], elem3[25], elem4[25];
int NEffElem = 0;          // Nilai efektif dari senarai kombinasi
// 2 buah senarai untuk menyimpan solusi dan solusi unik
string solution[300];
string solutionunique[300];
// Nilai efektif dari kedua senarai diatas
int NEffSol = 0;
int NEffSolUni = 0;
```

Gambar 2.2.2. Inisialisasi variabel global untuk mempermudah proses penampungan nilai elemen.

3. Prosedur randomNumber

```
void randomNumber (int* nums)
// Membangkitkan angka secara anak (random number generator)
// Seeds yang digunakan adalah waktu saat ini, sehingga relatif sulit terulang kembali
{
    // KAMUS LOKAL
    int i;
    int num;

    // ALGORITMA
    num = (rand() % 13) + 1;
    *nums = num;
}
```

Gambar 2.2.3. Prosedur randomNumber untuk membangkitkan bilangan secara acak.

4. Prosedur generate4

```
void generate4 (int* card1, int* card2, int* card3, int* card4)
// Mengenerate 4 angka random
// Menggunakan prosedur randomNumber yang telah diinisiasi sebelumnya
{
    // KAMUS LOKAL

    // ALGORITMA
    randomNumber(nums: card1);
    randomNumber(nums: card2);
    randomNumber(nums: card3);
    randomNumber(nums: card4);
}
```

Gambar 2.2.4. Prosedur generate4 untuk membangkitkan 4 buah bilangan secara acak.

5. Fungsi checkNumber

```
bool checkNumber (string str)
// Melakukan pengecekan apakah string yang masuk benar merupakan string atau integer
// Untuk memudahkan perlakuan konversi string ke integer
{
    // KAMUS LOKAL
    int i;

    // ALGORITMA
    for (i = 0; i <= str.length(); i++) {
        if (!isdigit(str[i])) {
            return false;
        } else {
            return true;
        }
    }
}
```

Gambar 2.2.5. Fungsi checkNumber untuk melakukan pengecekan apakah sebuah string dari masukan benar merupakan string atau integer dalam bentuk string.

6. Fungsi inputNumberValid

```
bool inputNumberValid (string input)
// Melakukan konversi dan validasi input berupa string ke integer
// Apakah valid (bernilai antara 2 - 10)
{
    // KAMUS LOKAL
    int result;

    // ALGORITMA
    result = std::stoi(str: input);
    if (result >= 2 && result <= 10) {
        return true;
    } else {
        return false;
    }
}
```

Gambar 2.2.6. Fungsi inputNumberValid untuk melakukan konversi dari string yang merupakan integer ke tipe data integer untuk divalidasi, apakah berada pada rentang 2 hingga 10.

7. Fungsi inputCharValid

```
bool inputCharValid (string input)
// Melakukan analisis elemen pertama dari sebuah string
// Apakah valid (bernilai A, K, Q, atau J)
{
    // KAMUS LOKAL
    char c[input.length()];
    int i=0;

    // ALGORITMA
    // Instansiasi for loop
    while (i < input.length()) {
        c[i] = input[i];
        i++;
    }

    // Validasi input
    if ((c[0] == 'A' || c[0] == 'K' || c[0] == 'Q' || c[0] == 'J') && input.length() == 1) {
        return true;
    } else {
        return false;
    }
}
```

Gambar 2.2.7. Fungsi inputCharValid untuk melakukan validasi pada masukan string, apakah string masukan memiliki panjang 1 dan bernilai A, K, Q, atau J.

8. Fungsi chartoInt

```
int chartoInt (string input)
// Melakukan konversi dari karakter karater yang mungkin dalam sebuah kartu menjadi sebuah integer
{
    // KAMUS LOKAL

    // ALGORITMA
    switch (input[0]) {
        case 'A' : return 1; break;
        case 'J' : return 11; break;
        case 'Q' : return 12; break;
        case 'K' : return 13; break;
    }
}
```

Gambar 2.2.8. Fungsi chartoInt untuk melakukan konversi dari karakter yang mungkin dalam kartu menjadi sebuah integer.

9. Fungsi inttoChar

```
void inttoChar (int input, string *target)
// Melakukan konversi dari integer yang mungkin dalam sebuah kartu menjadi karakter kartu yang bersesuaian
{
    // KAMUS LOKAL

    // ALGORITMA
    if (input >= 2 && input <= 10) {
        *target = to_string(val: input);
    } else {
        switch (input) {
            case 1  : *target = 'A'; break;
            case 11 : *target = 'J'; break;
            case 12 : *target = 'Q'; break;
            case 13 : *target = 'K'; break;
        }
    }
}
```

Gambar 2.2.9. Fungsi inttoChar untuk melakukan konversi dari integer yang mungkin dalam sebuah kartu menjadi karakter kartu yang bersesuaian.

10. Fungsi operations

```
char operations (int ops)
// Sebelumnya membuat operationsasi dari operasi dalam bentuk angka
// Fungsi ini mengembalikan bentuk angka dalam operasi menjadi operasi + - * /
{
    // KAMUS LOKAL

    // ALGORITMA
    switch (ops) {
        case 1 : return ('+');
        case 2 : return ('-');
        case 3 : return ('*');
        case 4 : return ('/');
    }
}
```

Gambar 2.2.10. Fungsi operations untuk mengembalikan bentuk dari integer ke operator dalam string.

11. Fungsi calculate

```
float calculate (int ops, float x, float y)
// Melakukan operasi bilangan real antara dua bilangan yang operationsasi nya diubah
// menjadi bilangan real (untuk memudahkan real division) dan operator dalam angka
{
    // KAMUS LOKAL

    // ALGORITMA
    switch (ops) {
        case 1 : return (x + y);
        case 2 : return (x - y);
        case 3 : return (x * y);
        case 4 : return ((float) x / (float) y);
    }
}
```

Gambar 2.2.11. Fungsi calculate untuk melakukan operasi sesuai dengan angka representasinya.

12. Prosedur checkCombination

```
void checkCombination (int a, int b, int c, int d)
// Melakukan pengecekan terhadap semua kombinasi yang mungkin dari masukan a, b, c, d
// Kombinasi yang sudah ada tidak dimasukkan ke dalam senarai
// Adapun kombinasi yang belum ada akan dimasukkan ke dalam senarai analisis
{
    // KAMUS LOKAL
    int i;
    bool found;

    // ALGORITMA
    // Inisialisasi
    i = 1, found= false;

    while (i < NEffElem && not found) {
        if (a == elem1[i] && b == elem2[i] && c == elem3[i] && d == elem4[i]) {
            // Jika masing-masing elemen yang bersesuaian sama, sudah ada dalam senarai
            found = true;
        } else {
            // Jika tidak maka akan dilakukan increment hingga ditemukan
            i++;
        }
    }
    // Berhasil keluar dari kalang, kondisi yang mungkin adalah i = NEffElem atau not found
    // Jika not found, maka akan nilai tersebut akan dimasukkan ke dalam senarai
    if (not found) {
        NEffElem++;
        elem1[NEffElem] = a; elem2[NEffElem] = b; elem3[NEffElem] = c; elem4[NEffElem] = d;
    }
}
```

Gambar 2.2.12. Prosedur checkCombination untuk melakukan pengecekan terhadap semua kombinasi yang mungkin dari masukan a, b, c, dan d.

13. Prosedur processRes

```
void processRes (int p, int q, int r, int w, int x, int y, int z)
// Melakukan hasil kalkulasi dengan bilangan masukan w, x, y, z dan operator p, q, r
// Operator p, q, r dihasilkan melalui skema brute force operator yang mungkin untuk mencapai nilai 24
{
    // KAMUS LOKAL
    float res;
    string sol, bil1, bil2, bil3, bil4;
    // Konversi bilangan ke string
    inttoChar(input: w, target: &bil1);
    inttoChar(input: x, target: &bil2);
    inttoChar(input: y, target: &bil3);
    inttoChar(input: z, target: &bil4);

    // ALGORITMA
    // Penanganan terhadap kasus penyusunan dan kombinasi operator yang mungkin
    // ((w ops x) ops y) ops z
    res = calculate(ops: r, calculate(ops: q, calculate(ops: p, x: (float) w, y: (float) x), (float) y),
    y: (float) z);
    if (res > 23.99999 && res < 24.00001) { // menghandle floating point akibat real division
        // Membuat solusi dalam bentuk string dan memasukkannya ke dalam senarai solusi
        sol = "((" + bil1 + " " + operations(ops: p) + " " + bil2 + ") " +
        operations(ops: q) + " " + bil3 + ") " + operations(ops: r) + " " + bil4;
        NEffSol++;
        solution[NEffSol] = sol;
    }
```

Gambar 2.2.13. Prosedur processRes untuk melakukan pemrosesan angka dengan operator yang mungkin demi mencapai nilai 24 – bagian (1).

```
    // (w ops (x ops y)) ops z
    res = calculate(ops: r, calculate(ops: p, x: (float) w, calculate(ops: q, (float (x)), (float) y)),
    y: (float) z);
    if (res > 23.99999 && res < 24.00001) {
        sol = "(" + bil1 + " " + operations(ops: p) + " (" + bil2 + " " + operations(ops: q) +
        " " + bil3 + ")) " + operations(ops: r) + " " + bil4;
        NEffSol++;
        solution[NEffSol] = sol;
    }

    // w ops ((x ops y) ops z)
    res = calculate(ops: p, x: (float) w, calculate(ops: r, calculate(ops: q, (float) x, (float) y),
    y: float (z)));
    if (res > 23.99999 && res < 24.00001) {
        sol = bil1 + " " + operations(ops: p) + " ((" + bil2 + " " + operations(ops: q) + " " +
        bil3 + ") " + operations(ops: r) + " " + bil4 + ")";
        NEffSol++;
        solution[NEffSol] = sol;
    }

    // w ops (x ops (y ops z))
    res = calculate(ops: p, x: (float) w, calculate(ops: q, (float) x, calculate(ops: r, x: (float) y, y:
    (float) z)));
    if (res > 23.99999 && res < 24.00001) {
        sol = bil1 + " " + operations(ops: p) + " (" + bil2 + " " + operations(ops: q) + " (" +
        bil3 + " " + operations(ops: r) + " " + bil4 + "))";
        NEffSol++;
        solution[NEffSol] = sol;
    }
```

Gambar 2.2.14. Prosedur processRes – bagian (2).

```
    // (w ops x) ops (y ops z)
    res = calculate(ops: q, calculate(ops: p, x: (float) w, y: (float) x), calculate(ops: r, x: (float)
    y, y: (float) z));
    if (res > 23.99999 && res < 24.00001) {
        sol = "(" + bil1 + " " + operations(ops: p) + " " + bil2 + ") " +
        operations(ops: q) + " (" + bil3 + " " + operations(ops: r) + " " + bil4 + ")";
        NEffSol++;
        solution[NEffSol] = sol;
    }
}
```

Gambar 2.2.15. Prosedur processRes – bagian (3).

14. Prosedur bruteForce

```
void bruteForce (int w, int x, int y, int z)
// Melakukan brute-force untuk menemukan pasangan solusi operator yang mungkin bagi bilangan
// w, x, y, dan z untuk mendapatkan nilai 24.
// Digunakan operator p, q, dan r dengan 3 kalang bersarang
{
    // KAMUS LOKAL
    int p, q, r;

    // ALGORITMA
    for (p = 1; p < 5; p++) {
        for (q = 1; q < 5; q++) {
            for (r = 1; r < 5; r++) {
                // Melakukan pemrosesan hasil untuk setiap operator yang mungkin
                processRes (p, q, r, w, x, y, z);
            }
        }
    }
}
```

Gambar 2.2.16. Prosedur bruteForce untuk menemukan seluruh pasangan solusi yang mungkin dari keempat bilangan masukan untuk mendapatkan nilai 24.

15. Prosedur input

```
void input ()
// Menangani proses input - masukan dari pengguna
// Pengguna dapat memilih untuk memasukkan input atau membangkitkan bilangan secara acak
{
    // KAMUS LOKAL
    string com;
    int pil;
    bool valid = false;

    // ALGORITMA
    cout << "====================  INPUT  ====================" << endl;
    while (!valid) {
        cout << "Tentukan cara membuat masukan" << endl;
        cout << " 1. Berdasarkan masukan dari pengguna" << endl;
        cout << " 2. Lakukan pembangkitan pasangan kartu acak" << endl;
        cout << "" << endl;
        cout << "Pilih mode" << endl << ">> ";
        cin >> com;
        if (checkNumber(str: com)) {
            pil = std::stoi(str: com);
            if (pil != 1 && pil != 2) {
                cout << "Masukan salah, ulangi!" << endl;
                cout << "--------------------------------------------------" << endl;
                valid = false;
            } else {
                valid = true;
            }
        } else {
            cout << "Masukan salah, ulangi!" << endl;
            cout << "--------------------------------------------------" << endl;
            valid = false;
        }
    }
```

Gambar 2.2.17. Prosedur input untuk melakukan penangan proses penerimaan masukan, baik melalui pengguna maupun pembangkitan bilangan secara acak – bagian (1).

```
// Keluar dari kalang, artinya input 1 atau 2
// Pemrosesan berdasarkan kondisi
if (pil == 1) {
    bool valid1 = false;
    bool valid2 = false;
    bool valid3 = false;
    bool valid4 = false;
    bool validful = valid1 && valid2 && valid3 && valid4;
    // Handling input lebih banyak dari 4 melalui analisis konversi panjang string
    char input4[100];
    string inputlast;
    cout << " " << endl;
    cout << "============  MASUKAN DARI PENGGUNA  ============" << endl;
    while (!validful) {
        cout << "Masukkan 4 buah kartu (A, 2-10, J, Q, atau K)" << endl << ">> ";
        cin >> p >> q >> r;
        // Skema berbeda saat meminta kartu keempat
        cin.getline(s: input4,n: 100);

        // Melakukan handling spacing di awal pada fungsi getline dengan mendefinisikan
        // array of character yang baru
        char* input5;
        input5 = (char*)malloc(100);
        for (int i = 0; input4[i] != '\0'; i++) {
            input5[i] = input4[i + 1];
        }
        // Melakukan konversi array of char ke string s untuk divalidasi
        s = input5;
```

Gambar 2.2.18. Prosedur input – bagian (2).

```
        // Skema validasi per input p, q, r, s
        // Validasi input p
        if (checkNumber(str: p)) {
            if (inputNumberValid(input: p)) {
                a = std::stoi(str: p);
                valid1 = true;
            } else {
                cout << "> Masukan angka pertama tidak valid" << endl;
                valid1 = false;
            }
        } else {
            if (inputCharValid(input: p)){
                a = chartoInt(input: p);
                valid1 = true;
            } else {
                cout << "> Masukan karakter pertama tidak valid" << endl;
                valid1 = false;
            }
        }
        // Validasi input q
        if (checkNumber(str: q)) {
            if (inputNumberValid(input: q)) {
                b = std::stoi(str: q);
                valid2 = true;
            } else {
                cout << "> Masukan angka kedua tidak valid" << endl;
                valid2 = false;
            }
```

Gambar 2.2.19. Prosedur input – bagian (3).

```
        } else {
            if (inputCharValid(input: q)){
                b = chartoInt(input: q);
                valid2 = true;
            } else {
                cout << "> Masukan karakter kedua tidak valid" << endl;
                valid2 = false;
            }
        }
        // Validasi input r
        if (checkNumber(str: r)) {
            if (inputNumberValid(input: r)) {
                c = std::stoi(str: r);
                valid3 = true;
            } else {
                cout << "> Masukan angka ketiga tidak valid" << endl;
                valid3 = false;
            }
        } else {
            if (inputCharValid(input: r)){
                c = chartoInt(input: r);
                valid3 = true;
            } else {
                cout << "> Masukan karakter ketiga tidak valid" << endl;
                valid3 = false;
            }
        }
        // Validasi input s
        // Validasi input berlebih yang ditangani di variabel terakhir
        if (s.length() > 2) {
            cout << "> Jumlah input lebih dari 4" << endl;
            valid4 = false;
```

Gambar 2.2.20. Prosedur input – bagian (4).

```
        } else {
            if (checkNumber(str: s)) {
                if (inputNumberValid(input: s)) {
                    d = std::stoi(str: s);
                    valid4 = true;
                } else {
                    cout << "> Masukan angka keempat tidak valid" << endl;
                    valid4 = false;
                }
            } else {
                if (inputCharValid(input: s)){
                    d = chartoInt(input: s);
                    valid4 = true;
                } else {
                    cout << "> Masukan karakter keempat tidak valid" << endl;
                    valid4 = false;
                }
            }
        }
        validful = valid1 && valid2 && valid3 && valid4;
        if (!validful) {
            cout << "Masukan salah, ulangi!" << endl;
            cout << "-----------------------------------------------" << endl;
        }
    }
    cout << "Cek hasil input :" << endl;
    cout << a << " " << b << " " << c << " " << d << endl;
```

Gambar 2.2.21. Prosedur input – bagian (5).

```
    } else {
        cout << " " << endl;
        cout << "===========  PEMBANGKITAN KARTU ACAK  ==========" << endl;
        cout << "Kartu hasil generate" << endl << ">> ";
        srand(time(0));
        generate4(card1: &a,card2: &b,card3: &c,card4: &d);
        // Pengisian karakter untuk masing-masing kartu
        inttoChar (input: a, target: &p);
        inttoChar (input: b, target: &q);
        inttoChar (input: c, target: &r);
        inttoChar (input: d, target: &s);
        cout << p << " " << q << " " << r << " " << s << endl;
    }
}
```

Gambar 2.2.22. Prosedur input – bagian (6).

16. Prosedur output

```
void output ()
// Menangani proses output - kembalian ke pengguna
// Pengguna dapat memilih untuk hanya menampilkan di terminak atau menyimpan ke file
{
    // KAMUS LOKAL
    string com, path, output;
    int pil;
    bool valid = false;

    // ALGORITMA
    cout << " " << endl;
    cout << "====================  OUTPUT  ====================" << endl;
    while (!valid) {
        cout << "Tentukan cara menerima hasil" << endl;
        cout << " 1. Keluarkan hasil pada terminal" << endl;
        cout << " 2. Simpan hasil ke dalam file" << endl;
        cout << "" << endl;
        cout << "Pilih mode" << endl << ">> ";
        cin >> com;
        if (checkNumber(str: com)) {
            pil = std::stoi(str: com);
            if (pil != 1 && pil != 2) {
                cout << "Masukan salah, ulangi!" << endl;
                cout << "-----------------------------------------------" << endl;
                valid = false;
            } else {
                valid = true;
            }
        } else {
            cout << "Masukan salah, ulangi!" << endl;
            cout << "-----------------------------------------------" << endl;
            valid = false;
        }
    }
```

Gambar 2.2.23. Prosedur output melakukan penangan proses pengembalian luaran, baik melalui terminal maupun penyimpanan dalam *file* – bagian (1).

```
// Keluar dari kalang, artinya input 1 atau 2
// Pemrosesan berdasarkan kondisi
if (pil == 1) {
    cout << " " << endl;
    cout << "==============  HASIL DI TERMINAL  ==============" << endl;
    cout << "Pasangan Kartu : " << p << " " << q << " " << r << " " << s << endl;
    // Handling tidak ada solusi
    if (NEffSolUni == 0) {
        cout << "Tidak terdapat solusi." << endl;
    } else {
        cout << "Terdapat " << NEffSolUni << " buah solusi" << endl;
        for (int i = 0; i <= NEffSolUni; i++) {
            cout << solutionunique[i] << endl;
        }
    }
    cout << endl;
```

Gambar 2.2.24. Prosedur output – bagian (2).

```
    } else {
        cout << " " << endl;
        cout << "================  HASIL DI FILE  ================" << endl;
        cout << "Masukkan path letak file txt akan disimpan" << endl << ">> test/";
        cin >> path;
        output = "test/" + path;
        ofstream MyFile(output);
        MyFile << "Pasangan Kartu : " << p << " " << q << " " << r << " " << s << endl;
        // Handling tidak ada solusi
        if (NEffSolUni == 0) {
            MyFile << "Tidak terdapat solusi.";
        } else {
            MyFile << "Terdapat " << NEffSolUni << " buah solusi" << endl;
            for (int i = 0; i < NEffSolUni; i++) {
                MyFile << solutionunique[i] << endl;
            }
            MyFile << solutionunique[NEffSolUni];
        }
        cout << "Hasil telah tersimpan dalam path yang telah diberikan" << endl;
        cout << endl;
    }
}
```

Gambar 2.2.25. Prosedur output – bagian (3).

17. Program utama

```
// Program Utama
int main() {
    system("CLS"); // clear screen
    cout << endl << "Tugas Kecil 1 - IF2211 Strategi Algoritma" << endl;
    cout << "Michael Leon Putra W / 13521108 / K-02" << endl;
    cout << "24 Card Game Solver" << endl;
    cout << " " << endl;

    // Pemanggilan prosedur input
    input();
```

Gambar 2.2.26. Program utama – bagian (1).

```
// Memasukkan semua kemungkinan pasangan bilangan untuk dicari menggunakan brute force
// Melakukan pemanggilan prosedur check combination, adapun terdapat 24 kemungkinan
checkCombination(a, b, c, d);
checkCombination(a, b, c: d, d: c);
checkCombination(a, b: c, c: b, d);
checkCombination(a, b: c, c: d, d: b);
checkCombination(a, b: d, c: b, d: c);
checkCombination(a, b: d, c, d: b);
checkCombination(a: b, b: a, c, d);
checkCombination(a: b, b: a, c: d, d: c);
checkCombination(a: b, b: c, c: a, d);
checkCombination(a: b, b: c, c: d, d: a);
checkCombination(a: b, b: d, c: a, d: c);
checkCombination(a: b, b: d, c, d: a);
checkCombination(a: c, b: a, c: b, d);
checkCombination(a: c, b: a, c: d, d: b);
checkCombination(a: c, b, c: a, d);
checkCombination(a: c, b, c: d, d: a);
checkCombination(a: c, b: d, c: a, d: b);
checkCombination(a: c, b: d, c: b, d: a);
checkCombination(a: d, b: a, c: b, d: c);
checkCombination(a: d, b: a, c, d: b);
checkCombination(a: d, b, c: a, d: c);
checkCombination(a: d, b, c, d: a);
checkCombination(a: d, b: c, c: a, d: b);
checkCombination(a: d, b: c, c: b, d: a);
```

Gambar 2.2.27. Program utama – bagian (2).

```
// Semua elemen yang dimasukkan ke dalam senarai diproses dengan berbagai operasi hingga
// ditemukan pasangan yang memenuhi
cout << " " << endl;
cout << "Mencari penyelesaian..." << endl;
cout << " " << endl;
// Proses pencarian solusi dimulai
auto std::chrono::...k::time_point start = high_resolution_clock::now();   // Menggunakan atribut
start pada jam
cout << "===============  HASIL EKSEKUSI  ================" << endl;
for (int i = 1; i <= NEffElem; i++) {
    bruteForce(w: elem1[i], x: elem2[i], y: elem3[i], z: elem4[i]);
}
// Proses pencarian solusi selesai
auto std::chrono::...k::time_point finish = high_resolution_clock::now();   // Menggunakan atribut
finish pada jam
```

Gambar 2.2.28. Program utama – bagian (3).

```
    // Unifikasi jawaban
    for (int i = 0; i <= NEffSol; i++) {
        for (int j = 0; j <= i; j++) {
            if (solution[i] == solution[j] && i != j) {
                break;
            } if (i == j && i != 0) {
                NEffSolUni++;
                solutionunique[NEffSolUni] = solution[i];
            }
        }
    }
    cout << "Pemrosesan selesai dilaksanakan" << endl;
    cout << "> Terdapat " << NEffSolUni << " buah solusi" << endl;

    // Pencetakan waktu eksekusi
    std::chrono::duration<double> diff = finish - start;
    cout << "> Waktu eksekusi: " << diff.count() << " sekon" << endl;

    // Outputting
    output();

    return 0;
}
```

Gambar 2.2.29. Program utama – bagian (4).

BAB III

PROGRAM TESTING

A. Tampilan Awal

Berikut adalah tampilan awal program yang telah dibuat.

```
Tugas Kecil 1 - IF2211 Strategi Algoritma
Michael Leon Putra W / 13521108 / K-02
24 Card Game Solver

====================  INPUT  ====================
Tentukan cara membuat masukan
 1. Berdasarkan masukan dari pengguna
 2. Lakukan pembangkitan pasangan kartu acak

Pilih mode
>>
```

Gambar 3.1. Tampilan awal program dan tampilan awal pemilihan menu input.

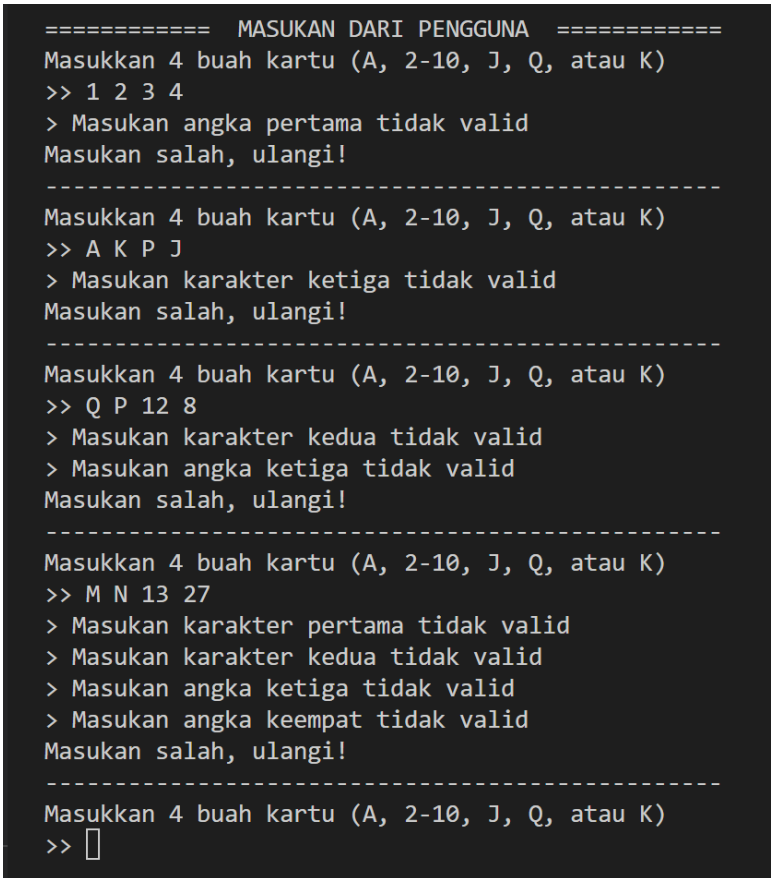
B. Skema Penerimaan Masukan

1. Masukan melalui pengguna
    a. Masukan tepat

```
====================  INPUT  ====================
Tentukan cara membuat masukan
 1. Berdasarkan masukan dari pengguna
 2. Lakukan pembangkitan pasangan kartu acak

Pilih mode
>> 1

============  MASUKAN DARI PENGGUNA  ============
Masukkan 4 buah kartu (A, 2-10, J, Q, atau K)
>> 2 3 4 5
Cek hasil input :
2 3 4 5

Mencari penyelesaian...
```

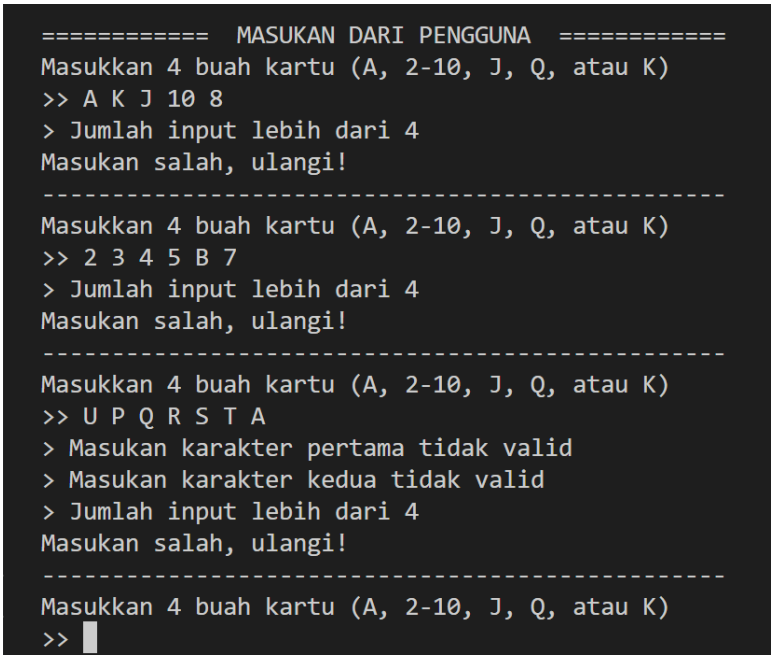
Gambar 3.2.1. Hasil tampilan jika masukan dari pengguna benar.

    b. Masukan tidak tepat - karakter dan angka



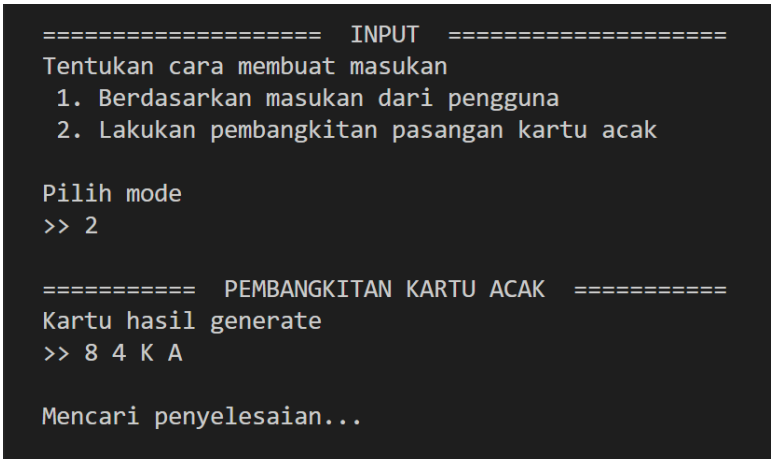
Gambar 3.2.2. Berbagai skema validasi masukan untuk masukan angka dan karakter.

c. Masukan tidak tepat - lebih dari 4



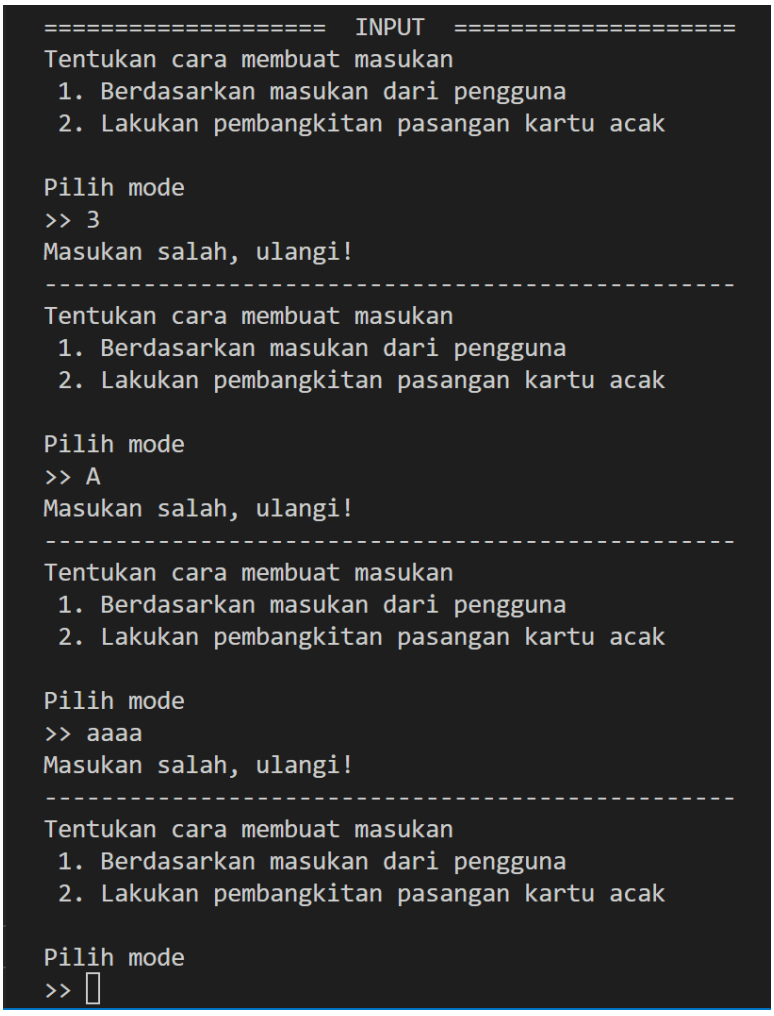
Gambar 3.2.3. Berbagai skema validasi masukan untuk masukan lebih dari 4.

2. Pembangkitan 4 buah kartu acak



Gambar 3.2.4. Hasil tampilan jika pengguna memilih untuk membangkitkan 4 buah kartu secara acak.

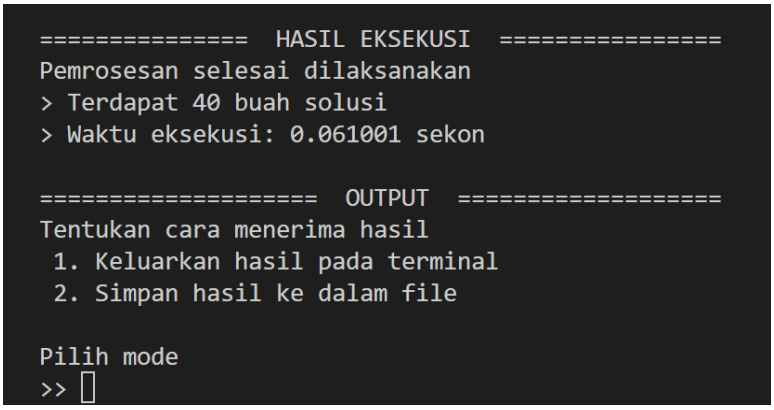
3. Skema validasi masukan untuk menu input



Gambar 3.2.5. Berbagai skema validasi masukan untuk menu input.

C. Skema Pemrosesan Solusi

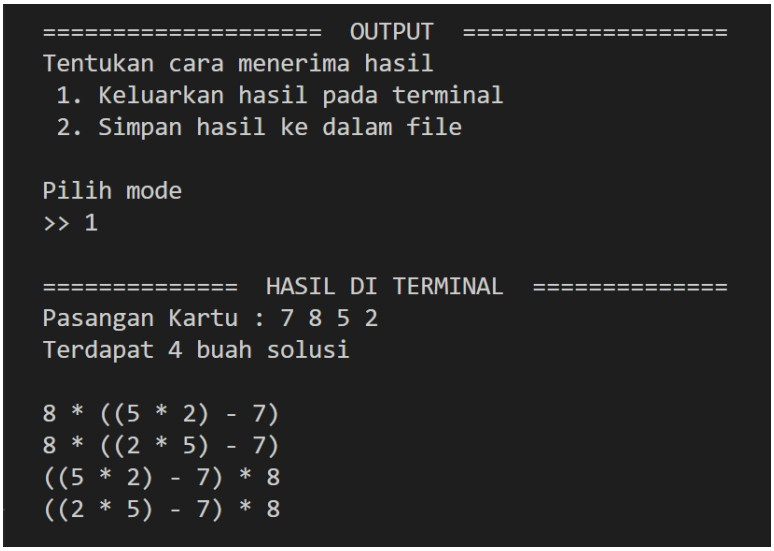
Berikut adalah tampilan setelah proses brute force selesai dilaksanakan.



Gambar 3.3. Tampilan jika proses eksekusi selesai dilaksanakan dan tampilan awal pemilihan menu output.

D. Skema Pengembalian Luaran

1. Pengembalian luaran melalui terminal



Gambar 3.4.1. Tampilan hasil pengembalian luaran di terminal.

2. Pengembalian luaran melalui *file*

```
====================  OUTPUT  ====================
Tentukan cara menerima hasil
 1. Keluarkan hasil pada terminal
 2. Simpan hasil ke dalam file

Pilih mode
>> 2

================  HASIL DI FILE  ================
Masukkan path letak file txt akan disimpan
>> test/hasil.txt
Hasil telah tersimpan dalam path yang telah diberikan
```

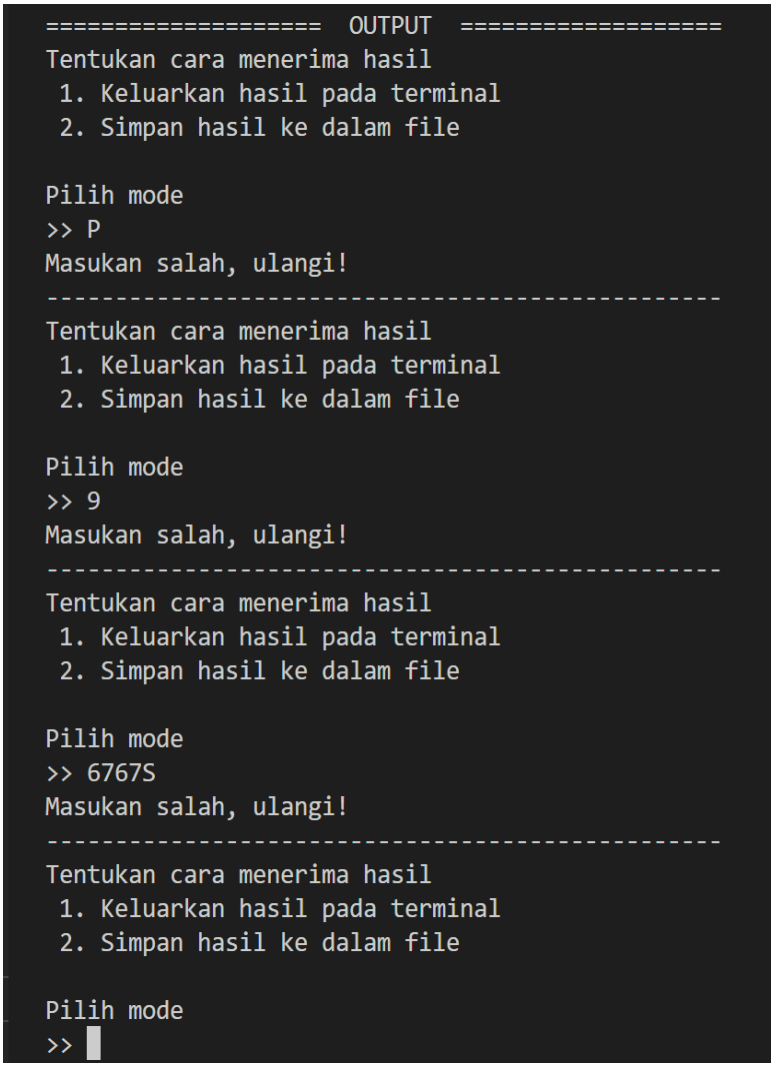
Gambar 3.4.2. Tampilan hasil pengembalian luaran di *file*.

test > hasil.txt

```
1   Pasangan Kartu : 7 8 5 2
2   Terdapat 4 buah solusi
3
4   8 * ((5 * 2) - 7)
5   8 * ((2 * 5) - 7)
6   ((5 * 2) - 7) * 8
7   ((2 * 5) - 7) * 8
```

Gambar 3.4.3. Isi dari *file* di dalam *path* yang dituju.

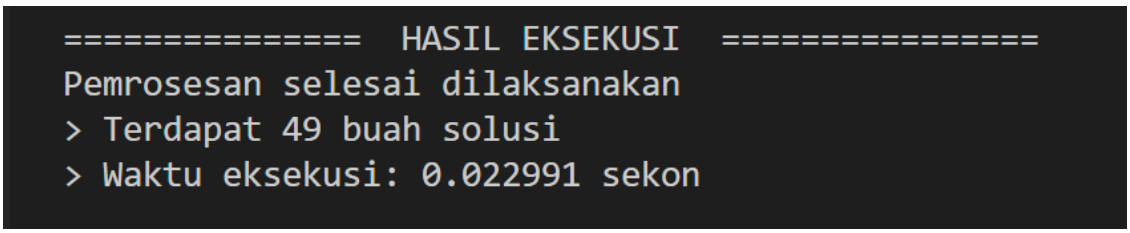
3. Skema validasi masukan untuk menu output



Gambar 3.4.4. Berbagai skema validasi masukan untuk menu output.

E. Test Case

1. Pasangan kartu : **A 8 9 Q**

Hasil Eksekusi

Solusi Penyelesaian

test > TestCase-(A89Q).txt

```
Pasangan Kartu : A 8 9 Q
Terdapat 49 buah solusi

((A - 8) + 9) * Q
(A - (8 - 9)) * Q
A * (8 * (Q - 9))
(A * 8) * (Q - 9)
((A + 9) - 8) * Q
(A + (9 - 8)) * Q
((A * Q) - 9) * 8
(A * (Q - 9)) * 8
A * ((Q - 9) * 8)
8 * ((A * Q) - 9)
8 * (A * (Q - 9))
(8 * A) * (Q - 9)
(8 / A) * (Q - 9)
8 / (A / (Q - 9))
8 * (Q - (A * 9))
8 * ((Q * A) - 9)
8 * ((Q / A) - 9)
(8 * (Q - 9)) * A
8 * ((Q - 9) * A)
8 * (Q - (9 * A))
(8 * (Q - 9)) / A
8 * ((Q - 9) / A)
8 * (Q - (9 / A))
```

test > TestCase-(A89Q).txt

```
8 / ((Q / 9) - A)
((9 + A) - 8) * Q
(9 + (A - 8)) * Q
((9 - 8) + A) * Q
(9 - (8 - A)) * Q
Q * ((A - 8) + 9)
Q * (A - (8 - 9))
(Q - (A * 9)) * 8
Q * ((A + 9) - 8)
Q * (A + (9 - 8))
((Q * A) - 9) * 8
((Q / A) - 9) * 8
((Q - 9) * A) * 8
(Q - (9 * A)) * 8
(Q - 9) * (A * 8)
((Q - 9) / A) * 8
(Q - (9 / A)) * 8
(Q - 9) / (A / 8)
Q * ((9 + A) - 8)
Q * (9 + (A - 8))
((Q - 9) * 8) * A
(Q - 9) * (8 * A)
((Q - 9) * 8) / A
(Q - 9) * (8 / A)
Q * ((9 - 8) + A)
Q * (9 - (8 - A))
```

2. Pasangan kartu : **2 3 4 5**

Hasil Eksekusi

`==========  HASIL EKSEKUSI  ==========`
`Pemrosesan selesai dilaksanakan`
`> Terdapat 40 buah solusi`
`> Waktu eksekusi: 0.040998 sekon`

Solusi Penyelesaian

test > TestCase-(2345).txt

```
Pasangan Kartu : 2 3 4 5
Terdapat 40 buah solusi

2 * ((3 + 4) + 5)
2 * (3 + (4 + 5))
2 * ((3 + 5) + 4)
2 * (3 + (5 + 4))
2 * ((4 + 3) + 5)
2 * (4 + (3 + 5))
2 * ((4 + 5) + 3)
2 * (4 + (5 + 3))
2 * ((5 + 3) + 4)
2 * (5 + (3 + 4))
2 * ((5 + 4) + 3)
2 * (5 + (4 + 3))
((3 - 2) + 5) * 4
(3 - (2 - 5)) * 4
((3 + 4) + 5) * 2
(3 + (4 + 5)) * 2
((3 + 5) - 2) * 4
(3 + (5 - 2)) * 4
((3 + 5) + 4) * 2
```

test > TestCase-(2345).txt

```
(3 + (5 + 4)) * 2
4 * ((3 - 2) + 5)
4 * (3 - (2 - 5))
((4 + 3) + 5) * 2
(4 + (3 + 5)) * 2
4 * ((3 + 5) - 2)
4 * (3 + (5 - 2))
4 * ((5 - 2) + 3)
4 * (5 - (2 - 3))
((4 + 5) + 3) * 2
(4 + (5 + 3)) * 2
4 * ((5 + 3) - 2)
4 * (5 + (3 - 2))
((5 - 2) + 3) * 4
(5 - (2 - 3)) * 4
((5 + 3) - 2) * 4
(5 + (3 - 2)) * 4
((5 + 3) + 4) * 2
(5 + (3 + 4)) * 2
((5 + 4) + 3) * 2
(5 + (4 + 3)) * 2
```

3. Pasangan kartu : **9 4 3 2**

Hasil Eksekusi

`==========  HASIL EKSEKUSI  ==========`
`Pemrosesan selesai dilaksanakan`
`> Terdapat 110 buah solusi`
`> Waktu eksekusi: 0.039963 sekon`

Solusi Penyelesaian

test > TestCase-(9432).txt

```
Pasangan Kartu : 9 4 3 2
Terdapat 110 buah solusi

((9 * 4) / 3) * 2
(9 * (4 / 3)) * 2
9 * ((4 / 3) * 2)
9 * (4 / (3 / 2))
(9 * 4) / (3 / 2)
((9 * 4) * 2) / 3
(9 * (4 * 2)) / 3
9 * ((4 * 2) / 3)
9 * (4 * (2 / 3))
(9 * 4) * (2 / 3)
(9 + 3) * (4 - 2)
((9 + 3) * 4) / 2
(9 + 3) * (4 / 2)
((9 / 3) * 4) * 2
(9 / 3) * (4 * 2)
(9 / (3 / 4)) * 2
9 / (3 / (4 * 2))
9 / ((3 / 4) / 2)
((9 + 3) / 2) * 4
(9 + 3) / (2 / 4)
((9 / 3) * 2) * 4
(9 / 3) * (2 * 4)
(9 / (3 / 2)) * 4
9 / (3 / (2 * 4))
9 / ((3 / 2) / 4)
((9 * 2) * 4) / 3
(9 * (2 * 4)) / 3
9 * ((2 * 4) / 3)
9 * (2 * (4 / 3))
(9 * 2) * (4 / 3)
((9 * 2) / 3) * 4
(9 * (2 / 3)) * 4
9 * ((2 / 3) * 4)
9 * (2 / (3 / 4))
(9 * 2) / (3 / 4)
(4 * (9 + 3)) / 2
4 * ((9 + 3) / 2)
((4 * 9) / 3) * 2
(4 * (9 / 3)) * 2
4 * ((9 / 3) * 2)
4 * (9 / (3 / 2))
(4 * 9) / (3 / 2)
((4 * 9) * 2) / 3
(4 * (9 * 2)) / 3
```

test > TestCase-(9432).txt

```
((4 / 3) * 2) * 9
(4 / 3) * (2 * 9)
(4 / (3 / 2)) * 9
4 / (3 / (2 * 9))
4 / ((3 / 2) / 9)
(4 - 2) * (9 + 3)
((4 * 2) * 9) / 3
(4 * (2 * 9)) / 3
4 * ((2 * 9) / 3)
4 * (2 * (9 / 3))
(4 * 2) * (9 / 3)
(4 / 2) * (9 + 3)
4 / (2 / (9 + 3))
(4 - 2) * (3 + 9)
((4 * 2) / 3) * 9
(4 * (2 / 3)) * 9
4 * ((2 / 3) * 9)
4 * (2 / (3 / 9))
(4 * 2) / (3 / 9)
(4 / 2) * (3 + 9)
4 / (2 / (3 + 9))
(3 + 9) * (4 - 2)
((3 + 9) * 4) / 2
(3 + 9) * (4 / 2)
((3 + 9) / 2) * 4
(3 + 9) / (2 / 4)
((2 * 9) * 4) / 3
(2 * (9 * 4)) / 3
2 * ((9 * 4) / 3)
2 * (9 * (4 / 3))
(2 * 9) * (4 / 3)
((2 * 9) / 3) * 4
(2 * (9 / 3)) * 4
2 * ((9 / 3) * 4)
2 * (9 / (3 / 4))
(2 * 9) / (3 / 4)
((2 * 4) * 9) / 3
(2 * (4 * 9)) / 3
2 * ((4 * 9) / 3)
2 * (4 * (9 / 3))
(2 * 4) * (9 / 3)
((2 * 4) / 3) * 9
(2 * (4 / 3)) * 9
2 * ((4 / 3) * 9)
2 * (4 / (3 / 9))
(2 * 4) / (3 / 9)
```

```
48   4 * ((9 * 2) / 3)
49   4 * (9 * (2 / 3))
50   (4 * 9) * (2 / 3)
51   (4 * (3 + 9)) / 2
52   4 * ((3 + 9) / 2)
53   ((4 / 3) * 9) * 2
54   (4 / 3) * (9 * 2)
55   (4 / (3 / 9)) * 2
56   4 / (3 / (9 * 2))
57   4 / ((3 / 9) / 2)
```

```
104   ((2 / 3) * 9) * 4
105   (2 / 3) * (9 * 4)
106   (2 / (3 / 9)) * 4
107   2 / (3 / (9 * 4))
108   2 / ((3 / 9) / 4)
109   ((2 / 3) * 4) * 9
110   (2 / 3) * (4 * 9)
111   (2 / (3 / 4)) * 9
112   2 / (3 / (4 * 9))
113   2 / ((3 / 4) / 9)
```

4. Pasangan kartu : **A K Q J**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 32 buah solusi
> Waktu eksekusi: 0.002001 sekon
```

Solusi Penyelesaian

test > TestCase-(AKQJ).txt

```
1   Pasangan Kartu : A K Q J
2   Terdapat 32 buah solusi
3
4   ((A * K) - J) * Q
5   (A * (K - J)) * Q
6   A * ((K - J) * Q)
7   A * (Q * (K - J))
8   (A * Q) * (K - J)
9   (K - (A * J)) * Q
10  ((K * A) - J) * Q
11  ((K / A) - J) * Q
12  ((K - J) * A) * Q
13  (K - (J * A)) * Q
14  (K - J) * (A * Q)
15  ((K - J) / A) * Q
16  (K - (J / A)) * Q
17  (K - J) / (A / Q)
18  ((K - J) * Q) * A
```

test > TestCase-(AKQJ).txt

```
19  (K - J) * (Q * A)
20  ((K - J) * Q) / A
21  (K - J) * (Q / A)
22  Q * ((A * K) - J)
23  Q * (A * (K - J))
24  (Q * A) * (K - J)
25  (Q / A) * (K - J)
26  Q / (A / (K - J))
27  Q * (K - (A * J))
28  Q * ((K * A) - J)
29  Q * ((K / A) - J)
30  (Q * (K - J)) * A
31  Q * ((K - J) * A)
32  Q * (K - (J * A))
33  (Q * (K - J)) / A
34  Q * ((K - J) / A)
35  Q * (K - (J / A))
```

5. Pasangan kartu : **6 6 6 6**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 7 buah solusi
> Waktu eksekusi: 0.003997 sekon
```

Solusi Penyelesaian

test > TestCase-(6666).txt

```
1   Pasangan Kartu : 6 6 6 6
2   Terdapat 7 buah solusi
3
4   ((6 + 6) + 6) + 6
5   (6 + (6 + 6)) + 6
6   6 + ((6 + 6) + 6)
7   6 + (6 + (6 + 6))
8   (6 + 6) + (6 + 6)
9   (6 * 6) - (6 + 6)
10  ((6 * 6) - 6) - 6
```

6. Pasangan kartu : **10 2 9 10**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 12 buah solusi
> Waktu eksekusi: 0.028001 sekon
```

Solusi Penyelesaian

test > TestCase-(102910).txt

```
1   Pasangan Kartu : 10 2 9 10
2   Terdapat 12 buah solusi
3
4   ((10 / 2) + 9) + 10
5   (10 / 2) + (9 + 10)
6   ((10 / 2) + 10) + 9
7   (10 / 2) + (10 + 9)
8   10 + (9 + (10 / 2))
```

test > TestCase-(102910).txt

```
9   (10 + 9) + (10 / 2)
10  (10 + (10 / 2)) + 9
11  10 + ((10 / 2) + 9)
12  (9 + (10 / 2)) + 10
13  9 + ((10 / 2) + 10)
14  9 + (10 + (10 / 2))
15  (9 + 10) + (10 / 2)
```

7. Pasangan kartu : **3 2 2 2**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 32 buah solusi
> Waktu eksekusi: 0.011038 sekon
```

Solusi Penyelesaian

test > TestCase-(3222).txt

```
1   Pasangan Kartu : 3 2 2 2
2   Terdapat 32 buah solusi
3
4   (3 * (2 + 2)) * 2
5   3 * ((2 + 2) * 2)
6   3 * (2 * (2 + 2))
7   (3 * 2) * (2 + 2)
8   ((3 * 2) * 2) * 2
9   (3 * (2 * 2)) * 2
10  3 * ((2 * 2) * 2)
11  3 * (2 * (2 * 2))
12  (3 * 2) * (2 * 2)
13  2 * (3 * (2 + 2))
14  (2 * 3) * (2 + 2)
15  ((2 * 3) * 2) * 2
16  (2 * (3 * 2)) * 2
17  2 * ((3 * 2) * 2)
18  2 * (3 * (2 * 2))
```

test > TestCase-(3222).txt

```
19  (2 * 3) * (2 * 2)
20  ((2 + 2) * 3) * 2
21  (2 + 2) * (3 * 2)
22  ((2 * 2) * 3) * 2
23  (2 * (2 * 3)) * 2
24  2 * ((2 * 3) * 2)
25  2 * (2 * (3 * 2))
26  (2 * 2) * (3 * 2)
27  ((2 + 2) * 2) * 3
28  (2 + 2) * (2 * 3)
29  (2 * (2 + 2)) * 3
30  2 * ((2 + 2) * 3)
31  ((2 * 2) * 2) * 3
32  (2 * (2 * 2)) * 3
33  2 * ((2 * 2) * 3)
34  2 * (2 * (2 * 3))
35  (2 * 2) * (2 * 3)
```

8. Pasangan kartu : **J 9 7 K**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 0 buah solusi
> Waktu eksekusi: 0.024992 sekon
```

Solusi Penyelesaian

test > TestCase-(J97K).txt

```
1   Pasangan Kartu : J 9 7 K
2   Tidak terdapat solusi.
```

9. Pasangan kartu : **2 4 3 A**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 242 buah solusi
> Waktu eksekusi: 0.039005 sekon
```

Solusi Penyelesaian

test > TestCase-(243A).txt

```
1   Pasangan Kartu : 2 4 3 A
2   Terdapat 242 buah solusi
3
4   (2 + 4) * (3 + A)
5   ((2 * 4) * 3) * A
6   (2 * (4 * 3)) * A
7   2 * ((4 * 3) * A)
8   2 * (4 * (3 * A))
9   (2 * 4) * (3 * A)
10  ((2 * 4) * 3) / A
11  (2 * (4 * 3)) / A
12  2 * ((4 * 3) / A)
13  2 * (4 * (3 / A))
14  (2 * 4) * (3 / A)
15  (2 + 4) * (A + 3)
```

test > TestCase-(243A).txt

```
124   4 * ((A * 2) * 3)
125   4 * (A * (2 * 3))
126   (4 * A) * (2 * 3)
127   ((4 / A) * 2) * 3
128   (4 / A) * (2 * 3)
129   (4 / (A / 2)) * 3
130   4 / (A / (2 * 3))
131   4 / ((A / 2) / 3)
132   4 * ((A + 3) + 2)
133   4 * (A + (3 + 2))
134   ((4 * A) * 3) * 2
135   (4 * (A * 3)) * 2
136   4 * ((A * 3) * 2)
137   4 * (A * (3 * 2))
138   (4 * A) * (3 * 2)
```

```
((2 * 4) * A) * 3
(2 * (4 * A)) * 3
2 * ((4 * A) * 3)
2 * (4 * (A * 3))
(2 * 4) * (A * 3)
((2 * 4) / A) * 3
(2 * (4 / A)) * 3
2 * ((4 / A) * 3)
2 * (4 / (A / 3))
(2 * 4) / (A / 3)
((2 * 3) * 4) * A
(2 * (3 * 4)) * A
2 * ((3 * 4) * A)
2 * (3 * (4 * A))
(2 * 3) * (4 * A)
((2 * 3) * 4) / A
(2 * (3 * 4)) / A
2 * ((3 * 4) / A)
2 * (3 * (4 / A))
(2 * 3) * (4 / A)
((2 + 3) + A) * 4
(2 + (3 + A)) * 4
((2 * 3) * A) * 4
(2 * (3 * A)) * 4
2 * ((3 * A) * 4)
2 * (3 * (A * 4))
(2 * 3) * (A * 4)
((2 * 3) / A) * 4
(2 * (3 / A)) * 4
2 * ((3 / A) * 4)
2 * (3 / (A / 4))
(2 * 3) / (A / 4)
((2 * A) * 4) * 3
(2 * (A * 4)) * 3
2 * ((A * 4) * 3)
2 * (A * (4 * 3))
(2 * A) * (4 * 3)
((2 / A) * 4) * 3
(2 / A) * (4 * 3)
(2 / (A / 4)) * 3
2 / (A / (4 * 3))
2 / ((A / 4) / 3)
((2 + A) + 3) * 4
(2 + (A + 3)) * 4
((2 * A) * 3) * 4
(2 * (A * 3)) * 4
```

```
((4 / A) * 3) * 2
(4 / A) * (3 * 2)
(4 / (A / 3)) * 2
4 / (A / (3 * 2))
4 / ((A / 3) / 2)
((3 * 2) * 4) * A
(3 * (2 * 4)) * A
3 * ((2 * 4) * A)
3 * (2 * (4 * A))
(3 * 2) * (4 * A)
((3 * 2) * 4) / A
(3 * (2 * 4)) / A
3 * ((2 * 4) / A)
3 * (2 * (4 / A))
(3 * 2) * (4 / A)
((3 + 2) + A) * 4
(3 + (2 + A)) * 4
((3 * 2) * A) * 4
(3 * (2 * A)) * 4
3 * ((2 * A) * 4)
3 * (2 * (A * 4))
(3 * 2) * (A * 4)
((3 * 2) / A) * 4
(3 * (2 / A)) * 4
3 * ((2 / A) * 4)
3 * (2 / (A / 4))
(3 * 2) / (A / 4)
((3 * 4) * 2) * A
(3 * (4 * 2)) * A
3 * ((4 * 2) * A)
3 * (4 * (2 * A))
(3 * 4) * (2 * A)
((3 * 4) * 2) / A
(3 * (4 * 2)) / A
3 * ((4 * 2) / A)
3 * (4 * (2 / A))
(3 * 4) * (2 / A)
((3 * 4) * A) * 2
(3 * (4 * A)) * 2
3 * ((4 * A) * 2)
3 * (4 * (A * 2))
(3 * 4) * (A * 2)
((3 * 4) / A) * 2
(3 * (4 / A)) * 2
3 * ((4 / A) * 2)
3 * (4 / (A / 2))
```

```
2 * ((A * 3) * 4)
2 * (A * (3 * 4))
(2 * A) * (3 * 4)
((2 / A) * 3) * 4
(2 / A) * (3 * 4)
(2 / (A / 3)) * 4
2 / (A / (3 * 4))
2 / ((A / 3) / 4)
(4 + 2) * (3 + A)
4 * ((2 + 3) + A)
4 * (2 + (3 + A))
((4 * 2) * 3) * A
(4 * (2 * 3)) * A
4 * ((2 * 3) * A)
4 * (2 * (3 * A))
(4 * 2) * (3 * A)
((4 * 2) * 3) / A
(4 * (2 * 3)) / A
4 * ((2 * 3) / A)
4 * (2 * (3 / A))
(4 * 2) * (3 / A)
(4 + 2) * (A + 3)
4 * ((2 + A) + 3)
4 * (2 + (A + 3))
((4 * 2) * A) * 3
(4 * (2 * A)) * 3
4 * ((2 * A) * 3)
4 * (2 * (A * 3))
(4 * 2) * (A * 3)
((4 * 2) / A) * 3
(4 * (2 / A)) * 3
4 * ((2 / A) * 3)
4 * (2 / (A / 3))
(4 * 2) / (A / 3)
4 * ((3 + 2) + A)
4 * (3 + (2 + A))
((4 * 3) * 2) * A
(4 * (3 * 2)) * A
4 * ((3 * 2) * A)
4 * (3 * (2 * A))
(4 * 3) * (2 * A)
((4 * 3) * 2) / A
(4 * (3 * 2)) / A
4 * ((3 * 2) / A)
4 * (3 * (2 / A))
(4 * 3) * (2 / A)
```

```
(3 * 4) / (A / 2)
((3 + A) + 2) * 4
(3 + (A + 2)) * 4
(3 + A) * (2 + 4)
((3 * A) * 2) * 4
(3 * (A * 2)) * 4
3 * ((A * 2) * 4)
3 * (A * (2 * 4))
(3 * A) * (2 * 4)
((3 / A) * 2) * 4
(3 / A) * (2 * 4)
(3 / (A / 2)) * 4
3 / (A / (2 * 4))
3 / ((A / 2) / 4)
(3 + A) * (4 + 2)
((3 * A) * 4) * 2
(3 * (A * 4)) * 2
3 * ((A * 4) * 2)
3 * (A * (4 * 2))
(3 * A) * (4 * 2)
((3 / A) * 4) * 2
(3 / A) * (4 * 2)
(3 / (A / 4)) * 2
3 / (A / (4 * 2))
3 / ((A / 4) / 2)
((A * 2) * 4) * 3
(A * (2 * 4)) * 3
A * ((2 * 4) * 3)
A * (2 * (4 * 3))
(A * 2) * (4 * 3)
((A + 2) + 3) * 4
(A + (2 + 3)) * 4
((A * 2) * 3) * 4
(A * (2 * 3)) * 4
A * ((2 * 3) * 4)
A * (2 * (3 * 4))
(A * 2) * (3 * 4)
((A * 4) * 2) * 3
(A * (4 * 2)) * 3
A * ((4 * 2) * 3)
A * (4 * (2 * 3))
(A * 4) * (2 * 3)
((A * 4) * 3) * 2
(A * (4 * 3)) * 2
A * ((4 * 3) * 2)
A * (4 * (3 * 2))
```

```
108   4 * ((3 + A) + 2)
109   4 * (3 + (A + 2))
110   ((4 * 3) * A) * 2
111   (4 * (3 * A)) * 2
112   4 * ((3 * A) * 2)
113   4 * (3 * (A * 2))
114   (4 * 3) * (A * 2)
115   ((4 * 3) / A) * 2
116   (4 * (3 / A)) * 2
117   4 * ((3 / A) * 2)
118   4 * (3 / (A / 2))
119   (4 * 3) / (A / 2)
120   4 * ((A + 2) + 3)
121   4 * (A + (2 + 3))
122   ((4 * A) * 2) * 3
123   (4 * (A * 2)) * 3
```

```
231   (A * 4) * (3 * 2)
232   ((A + 3) + 2) * 4
233   (A + (3 + 2)) * 4
234   (A + 3) * (2 + 4)
235   ((A * 3) * 2) * 4
236   (A * (3 * 2)) * 4
237   A * ((3 * 2) * 4)
238   A * (3 * (2 * 4))
239   (A * 3) * (2 * 4)
240   (A + 3) * (4 + 2)
241   ((A * 3) * 4) * 2
242   (A * (3 * 4)) * 2
243   A * ((3 * 4) * 2)
244   A * (3 * (4 * 2))
245   (A * 3) * (4 * 2)
```

10. Pasangan kartu : **7 8 7 7**

Hasil Eksekusi

```
===============  HASIL EKSEKUSI  ================
Pemrosesan selesai dilaksanakan
> Terdapat 0 buah solusi
> Waktu eksekusi: 0.008969 sekon
```

Solusi Penyelesaian

test > TestCase-(7877).txt

```
1   Pasangan Kartu : 7 8 7 7
2   Tidak terdapat solusi.
```

BAB IV

LAMPIRAN

Berikut terlampir link *drive source code* serta isinya sesuai dengan spesifikasi :

https://drive.google.com/drive/folders/15bQ4F6u6gs1ZPz-i8cJbdMMWe3j3Qx9d?usp=sharing

Berikut Repository Github:

https://github.com/mikeleo03/Tucil1_13521108

Status Penyelesaian : *Completed*

No.	Poin	Ya	Tidak
1.	Program berhasil dikompilasi tanpa kesalahan	✓	
2.	Program berhasil *running*	✓	
3.	Program dapat membaca *input* / *generate* sendiri dan memberikan luaran	✓	
4.	Solusi yang diberikan program memenuhi (berhasil mencapai 24)	✓	
5.	Program dapat menyimpan solusi dalam *file* teks	✓	