TUGAS DESAIN ANALISIS ALGORITMA

DOKUMENTASI KODE 20 SOLVER

Disusun Oleh :

Muhammad Arif Amijoyo	(L0123089)
Muhammad Azka Ibrahim	(L0123091)

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA
UNIVERSITAS SEBELAS MARET

2024

A. Pembagian Tugas

Muhammad Arif Amijoyo : HTML, CSS, Demo Video
Muhammad Azka Ibrahim : JavaScript, Dokumentasi

B. Penjabaran dari Problem

Problem: Pengguna memberikan empat angka, tugas program adalah mencari semua ekspresi aritmatika yang menggunakan keempat angka tersebut bersama dengan operator dasar (+, -, *, /,()) sehingga hasil ekspresi tersebut adalah **20**. Jika ada solusi, cetak semua ekspresi yang valid. Jika tidak ada solusi, tampilkan pesan bahwa tidak mungkin menghasilkan 20 dengan angka dan operasi yang diberikan.

Input-Output:

- **Input:** Empat angka (misalnya: 2, 3, 5, 10).
- **Output:**
    - Jika ada solusi: Cetak semua ekspresi yang valid yang menghasilkan 20, misalnya: 10 * 3 - 5 * 2 = 20.
    - Jika tidak ada solusi: Cetak pesan "No valid Expressions."

Pendekatan Brute Force:

1. **Permutasi Angka:**
    - Buat semua urutan permutasi dari empat angka yang diberikan. Ini karena urutan angka mempengaruhi hasil operasi.
    - Contoh untuk angka 2, 3, 5, 10 ada 24 permutasi (4! = 24).
2. **Operator:**
    - Gunakan operator dasar aritmatika: +, -, *, dan /.
    - Kombinasikan operator dengan angka dalam setiap permutasi, sehingga ada banyak variasi kemungkinan operasi yang bisa dilakukan.
3. **Urutan Perhitungan:**

- Pertimbangkan penggunaan tanda kurung untuk mengubah urutan perhitungan. Urutan operasi sangat penting dalam aritmatika, dan penggunaan tanda kurung dapat memengaruhi hasil akhir.
- Contoh ekspresi dengan urutan berbeda:
  - 10 * 3 - 5 * 2
  - (10 * 3) - (5 * 2)
  - (10 * (3 - 5)) * 2

4. **Evaluasi Ekspresi:**
   - Setiap ekspresi yang dihasilkan dari permutasi angka dan operator perlu dievaluasi.
   - Jika hasil ekspresi tersebut adalah 20, simpan ekspresi itu sebagai solusi.
   - Hindari pembagian dengan nol karena ini menyebabkan kesalahan.
5. **Hasil Akhir:**
   - Jika ada ekspresi yang hasilnya 20, cetak ekspresi tersebut.
   - Jika setelah mencoba semua kemungkinan tidak ditemukan ekspresi yang valid, maka program mengeluarkan output "ini tidak mungkin."

C. Penjelasan dari Implementasi

Fungsi processInput():

```
function processInput() {
    const num1 = parseFloat(document.getElementById('number1').value);
    const num2 = parseFloat(document.getElementById('number2').value);
    const num3 = parseFloat(document.getElementById('number3').value);
    const num4 = parseFloat(document.getElementById('number4').value);

    const invalidMessage = document.getElementById('invalid');
    const outputElement = document.getElementById('output');
    const noValid = document.getElementById('noValid');

    noValid.innerHTML = '';
    invalidMessage.style.display = 'none';
    outputElement.innerHTML = '';

    if (num1 === '' || num2 === '' || num3 === '' || num4 === '') {
        invalidMessage.innerHTML = 'Please enter valid numbers.';
        invalidMessage.style.display = 'block';
        return;
    }

    if (isNaN(num1) || isNaN(num2) || isNaN(num3) || isNaN(num4)) {
        invalidMessage.innerHTML = 'Please enter valid numbers.';
        invalidMessage.style.display = 'block';
        return;
    }

    const numbers = [num1, num2, num3, num4];
    const validExpressions = generateExpressions(numbers);

    if (validExpressions.length > 0) {
        validExpressions.forEach(expr => {
            const p = document.createElement('p');
            p.textContent = expr;
            outputElement.appendChild(p);
        });
    } else {
        noValid.innerHTML = 'No valid expressions found';
    }
}
```

Fungsi ini menangani proses pengambilan input dari pengguna, validasi, dan memanggil fungsi lain untuk menghasilkan ekspresi matematika.

1. **Mengambil input pengguna dan langsung mengubahnya menjadi angka:**

```
const num1 = parseFloat(document.getElementById('number1').value);
const num2 = parseFloat(document.getElementById('number2').value);
const num3 = parseFloat(document.getElementById('number3').value);
const num4 = parseFloat(document.getElementById('number4').value);
```

Kode ini menggunakan parseFloat() untuk mengambil nilai dari elemen input HTML dan langsung mengonversinya menjadi angka. Jika input kosong atau tidak valid (misalnya, bukan angka), maka nilainya akan menjadi NaN.

2. **Mengambil elemen untuk pesan kesalahan dan hasil:**

```
const invalidMessage = document.getElementById('invalid');
const outputElement = document.getElementById('output');
const noValid = document.getElementById('noValid');
```

Elemen-elemen HTML yang digunakan untuk menampilkan pesan kesalahan (invalidMessage), hasil (outputElement), dan pesan jika tidak ada ekspresi valid (noValid) diambil.

3. **Menghapus pesan dan hasil sebelumnya:**

```
noValid.innerHTML = '';
invalidMessage.style.display = 'none';
outputElement.innerHTML = '';
```

Sebelum memproses input baru, hasil sebelumnya dihapus dari halaman untuk memastikan pengguna melihat hasil terbaru. Pesan kesalahan juga disembunyikan.

4. **Memvalidasi apakah input valid:**

```
if (num1 === '' || num2 === '' || num3 === '' || num4 === '') {
    invalidMessage.innerHTML = 'Please enter valid numbers.';
    invalidMessage.style.display = 'block';
    return;
}
```

Jika salah satu dari nilai yang diperoleh adalah NaN (yang bisa terjadi jika input kosong atau bukan angka valid), maka pesan kesalahan ditampilkan dan fungsi dihentikan.

5. **Membuat array angka dari input:**

```
const numbers = [num1, num2, num3, num4];
```

Setelah memastikan bahwa semua input adalah angka valid, angka-angka tersebut disimpan dalam array numbers.

6. **Menghasilkan ekspresi valid menggunakan generateExpressions():**

```
const validExpressions = generateExpressions(numbers);
```

Fungsi generateExpressions() dipanggil dengan array numbers untuk menghasilkan semua kombinasi ekspresi aritmatika yang mungkin dengan menggunakan angka-angka tersebut dan empat operator dasar (+, -, *, /).

7. **Menampilkan hasil:**

```
if (validExpressions.length > 0) {
    validExpressions.forEach(expr => {
        const p = document.createElement('p');
        p.textContent = expr;
        outputElement.appendChild(p);
    });
} else {
    noValid.innerHTML = 'No valid expressions found';
}
```

- Jika ada ekspresi valid yang menghasilkan nilai 20, ekspresi-ekspresi tersebut ditampilkan di elemen outputElement. Setiap ekspresi ditampilkan dalam elemen <p>.
- Jika tidak ada ekspresi valid yang ditemukan, pesan "No valid expressions found" ditampilkan.

Fungsi generateExpressions(numbers)

```
function generateExpressions(numbers) {
    const operators = ['+', '-', '*', '/'];
    const results = new Set();

    const permute = (arr) => {
        if (arr.length === 1) return [arr];
        let permutations = [];
        for (let i = 0; i < arr.length; i++) {
            const rest = arr.slice(0, i).concat(arr.slice(i + 1));
            const permutedRest = permute(rest);
            for (const perm of permutedRest) {
                permutations.push([arr[i], ...perm]);
            }
        }
        return permutations;
    };

    const evaluateExpression = (expr) => {
        try {
            return Function(`'use strict'; return (${expr})`)();
        } catch (e) {
            return null;
        }
    };

    const generateAllExpressions = (nums) => {
        let expressions = [];
        for (let op1 of operators) {
            for (let op2 of operators) {
                for (let op3 of operators) {
                    const exprs = [
                        `${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]} ${op3} ${nums[3]}`,
                        `(${nums[0]} ${op1} ${nums[1]}) ${op2} (${nums[2]} ${op3} ${nums[3]})`,
                        `(${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
                        `${nums[0]} ${op1} (${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
                        `${nums[0]} ${op1} (${nums[1]} ${op2} (${nums[2]} ${op3} ${nums[3]}))`
                    ];
                    expressions.push(...exprs);
                }
            }
        }
        return expressions;
    };

    const numbersPermutations = permute(numbers);
    for (const nums of numbersPermutations) {
        const expressions = generateAllExpressions(nums);
        for (const expr of expressions) {
            const result = evaluateExpression(expr);
            if (result === 20 && !expr.includes('/ 0')) {
                results.add(expr);
            }
        }
    }

    return Array.from(results);
}
```

```
const operators = ['+', '-', '*', '/'];
const results = new Set();
```

const operators = ['+', '-', '*', '/'];

Ini adalah sebuah array yang menyimpan empat operator aritmatika dasar:

- + (penjumlahan)
- - (pengurangan)
- * (perkalian)
- / (pembagian)

const results = new Set();

Ini mendeklarasikan sebuah variabel results sebagai instance dari Set, yaitu struktur data di JavaScript yang hanya menyimpan nilai unik, tidak ada elemen duplikat. Dalam konteks kode ini, results digunakan untuk menyimpan ekspresi matematika yang valid yang menghasilkan angka 20, memastikan bahwa tidak ada ekspresi yang sama disimpan lebih dari sekali.

```
const permute = (arr) => {
    if (arr.length === 1) return [arr];
    let permutations = [];
    for (let i = 0; i < arr.length; i++) {
        const rest = arr.slice(0, i).concat(arr.slice(i + 1));
        const permutedRest = permute(rest);
        for (const perm of permutedRest) {
            permutations.push([arr[i], ...perm]);
        }
    }
    return permutations;
};
```

Const permute = (arr) => {.....}

Fungsi ini bertugas untuk menghasilkan semua permute atau permutasi dari elemen-elemen dalam array.

1. **Base Case:**

   ```
   if (arr.length === 1) return [arr];
   ```

   Ini adalah kondisi dasar atau base case dari fungsi rekursif. Jika panjang array hanya 1, maka permutasi dari array tersebut hanyalah array itu sendiri. Fungsi akan berhenti melakukan rekursi di titik ini.

2. **Inisialisasi Array untuk Menyimpan Permutasi:**

   ```
   let permutations = []
   ```

   Sebuah array permutations diinisialisasi untuk menyimpan semua permutasi yang dihasilkan selama proses.

3. **Iterasi dan Pemilihan Elemen untuk Ditempatkan pada Posisi Awal:**

   ```
   For (let i = 0; i < arr.length; i++) {
       const rest = arr.slice(0, i).concat(arr.slice(i+1))
   ```

   Fungsi ini melakukan iterasi pada setiap elemen arr. Pada setiap iterasi, elemen ke-i dipilih sebagai elemen pertama dari permutasi, dan sisa elemen lainnya disimpan dalam array rest.

   Misalnya, jika arr = [1, 2, 3] dan i = 0, maka arr[i] = 1, dan rest = [2, 3].

4. **Rekursi untuk Menghasilkan Permutasi dari Elemen Sisanya:**

   ```
   const permutedRest = permute(rest)
   ```

   Fungsi kemudian memanggil dirinya sendiri (`permute`) dengan array `rest` untuk menghitung semua permutasi dari elemen-elemen selain elemen yang dipilih.

5. **Menggabungkan Elemen yang Dipilih dengan Permutasi dari Sisanya:**

   for (const perm of permutedRest) {

   permutations.push([arr[i], ...perm]);

   }

   Setelah semua permutasi dari elemen-elemen sisanya dihitung, fungsi menambahkan elemen yang dipilih (arr[i]) ke setiap permutasi dari rest dan memasukkannya ke dalam array permutations.

6. **Mengembalikan Semua Permutasi:**

   return permutations

   Setelah semua permutasi dari array arr dihitung, fungsi mengembalikan array permutations yang berisi semua kombinasi permutasi.

```
const evaluateExpression = (expr) => {
    try {
        return Function(`'use strict'; return (${expr})`)();
    } catch (e) {
        return null;
    }
};
```

Const evaluateExpression = (arr) => {.....}

Fungsi evaluateExpression bertanggung jawab untuk mengevaluasi ekspresi matematika yang diberikan dalam bentuk string dan mengembalikan hasilnya. Fungsi ini menggunakan metode dinamis untuk mengeksekusi kode JavaScript yang dihasilkan dari string ekspresi.

1. **Function Konstruktor:**

```
Function(`'use strict'; return (${expr})`)();
```

   - Di sini, Function adalah konstruktor bawaan JavaScript yang memungkinkan pembuatan fungsi baru dari string kode.
   - Fungsi ini dibuat dengan 'use strict' di dalamnya untuk menerapkan mode ketat JavaScript. Mode ketat membantu menghindari kesalahan umum, seperti mendefinisikan variabel tanpa kata kunci var, let, atau const.
   - String expr (ekspresi matematika dalam bentuk teks) dibungkus dalam return (...), di mana ekspresi akan dievaluasi, dan hasil dari ekspresi tersebut dikembalikan.

2. **try-catch Blok:**
   - Bagian try digunakan untuk menangani kode yang mungkin gagal saat dijalankan.
   - Jika terjadi kesalahan (misalnya, ekspresi tidak valid seperti pembagian dengan nol atau sintaksis yang salah), kode di dalam catch akan dijalankan.
   - Jika ada error, fungsi mengembalikan null sebagai hasil, yang berarti ekspresi tersebut tidak valid.

const generateAllExpressions

```
const generateAllExpressions = (nums) => {
    let expressions = [];
    for (let op1 of operators) {
        for (let op2 of operators) {
            for (let op3 of operators) {
                const exprs = [
                    `${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]} ${op3} ${nums[3]}`,
                    `(${nums[0]} ${op1} ${nums[1]}) ${op2} (${nums[2]} ${op3} ${nums[3]})`,
                    `(${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
                    `${nums[0]} ${op1} (${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
                    `${nums[0]} ${op1} (${nums[1]} ${op2} (${nums[2]} ${op3} ${nums[3]}))`
                ];
                expressions.push(...exprs);
            }
        }
    }
    return expressions;
};
```

Fungsi generateAllExpressions bertanggung jawab untuk menghasilkan semua ekspresi matematika yang mungkin dari kombinasi empat angka (nums) dengan operator aritmatika. Fungsi ini membentuk berbagai variasi ekspresi dengan menggunakan tanda kurung untuk menentukan urutan operasi yang berbeda.

1. Deklarasi Array expressions:

   let expressions = []

   Array ini digunakan untuk menyimpan semua ekspresi yang dihasilkan.
2. Tiga Lapisan Loop:
   - **op1**, **op2**, dan **op3** mewakili tiga operator yang digunakan dalam ekspresi, dipilih dari array operators yang sebelumnya didefinisikan sebagai ['+', '-', '*', '/'].
   - Loop bersarang digunakan untuk menghasilkan kombinasi operator yang berbeda untuk tiga posisi operator di antara empat angka.
3. Berbagai Bentuk Ekspresi yang Dihasilkan:

```
const exprs = [
    `${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]} ${op3} ${nums[3]}`,
    `(${nums[0]} ${op1} ${nums[1]}) ${op2} (${nums[2]} ${op3} ${nums[3]})`,
    `(${nums[0]} ${op1} ${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
    `${nums[0]} ${op1} (${nums[1]} ${op2} ${nums[2]}) ${op3} ${nums[3]}`,
    `${nums[0]} ${op1} (${nums[1]} ${op2} (${nums[2]} ${op3} ${nums[3]}))`
];
expressions.push(...exprs);
```

Beberapa variasi ekspresi dibuat dengan menambahkan tanda kurung untuk mengubah prioritas operasi:

- Ekspresi sederhana tanpa tanda kurung:
  Contoh: num1 op1 num2 op2 num3 op3 num4 → 1 + 2 * 3 / 4
- Ekspresi dengan dua grup:
  Contoh: (num1 op1 num2) op2 (num3 op3 num4) → (1 + 2) * (3 / 4)
- Ekspresi dengan tanda kurung di sekitar tiga angka:
  Contoh: (num1 op1 num2 op2 num3) op3 num4 → (1 + 2 * 3) / 4
- Ekspresi dengan tanda kurung di sekitar dua angka di tengah:
  Contoh: num1 op1 (num2 op2 num3) op3 num4 → 1 + (2 * 3) / 4
- Ekspresi dengan tanda kurung bertingkat:
  Contoh: num1 op1 (num2 op2 (num3 op3 num4)) → 1 + (2 * (3 / 4))

4. Menyimpan Ekspresi ke dalam Array expressions:

   `expressions.push(...exprs)`

   Setiap iterasi menghasilkan lima ekspresi berbeda, yang kemudian ditambahkan ke array expressions menggunakan spread operator (...), sehingga elemen-elemen array exprs dimasukkan ke dalam array expressions.

5. Mengembalikan Semua Ekspresi:

   `return expressions`

   Setelah semua kombinasi angka dan operator dievaluasi, fungsi mengembalikan array expressions yang berisi semua ekspresi matematika yang mungkin.

const numbersPermutations = permute(numbers)

```
const numbersPermutations = permute(numbers);
for (const nums of numbersPermutations) {
    const expressions = generateAllExpressions(nums);
    for (const expr of expressions) {
        const result = evaluateExpression(expr);
        if (result === 20 && !expr.includes('/ 0')) {
            results.add(expr);
        }
    }
}
```

Kode ini merupakan bagian dari keseluruhan algoritma yang berfungsi untuk mencari kombinasi angka dan operator yang menghasilkan nilai 20 dari empat angka yang diberikan.

1. Menghasilkan Semua Permutasi Angka:

   ```
   const numbersPermutations = permute(numbers);
   ```

   - Fungsi permute(numbers) digunakan untuk menghasilkan semua permutasi dari array numbers.
   - Misalkan numbers = [1, 2, 3, 4], maka numbersPermutations akan berisi semua kemungkinan urutan angka, seperti:

2. Iterasi Melalui Setiap Permutasi Angka:

   ```
   for (const nums of numbersPermutations) {
   ```

   Ini adalah loop yang akan berjalan melalui setiap permutasi angka (nums), di mana nums akan berisi urutan angka yang berbeda dalam setiap iterasi.

3. Menghasilkan Semua Ekspresi dari Setiap Permutasi:

   ```
   const expressions = generateAllExpressions(nums);
   ```

Fungsi generateAllExpressions(nums) digunakan untuk menghasilkan semua ekspresi matematika dari urutan angka tersebut (nums), dengan menempatkan operator di antara angka-angka dan menambahkan tanda kurung pada berbagai kombinasi yang memungkinkan.
Hasilnya adalah array expressions, yang berisi semua ekspresi yang mungkin untuk urutan angka tersebut.

4. Evaluasi Setiap Ekspresi:

```
for (const expr of expressions) {
    const result = evaluateExpression(expr);
```

   - Loop ini akan mengevaluasi setiap ekspresi (expr) dari array expressions yang dihasilkan sebelumnya.
   - Fungsi evaluateExpression(expr) digunakan untuk mengevaluasi nilai dari ekspresi tersebut.
   - Misalnya, jika expr = "1 + 2 * 3 - 4", maka evaluateExpression(expr) akan menghasilkan nilai **3**.

5. Memeriksa Hasil Evaluasi:

```
if (result === 20 && !expr.includes('/ 0')) {
```

   - Setelah ekspresi dievaluasi, hasilnya (result) diperiksa. Jika hasilnya **20**, ekspresi tersebut memenuhi tujuan dari algoritma (yaitu menghasilkan angka 20).
   - Selain itu, pengecekan dilakukan agar ekspresi tersebut **tidak mengandung pembagian dengan nol**. Jika ekspresi mengandung pembagian dengan nol ('/ 0'), ekspresi tersebut diabaikan karena akan menghasilkan kesalahan.

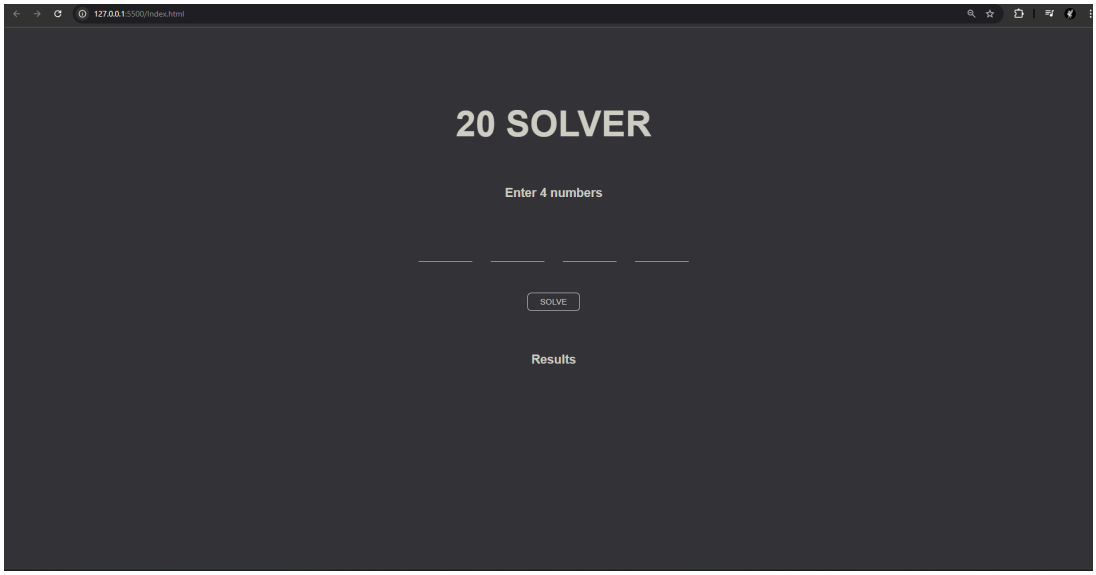
6. Menyimpan Ekspresi yang Valid:

```
return Array.from(results);
```

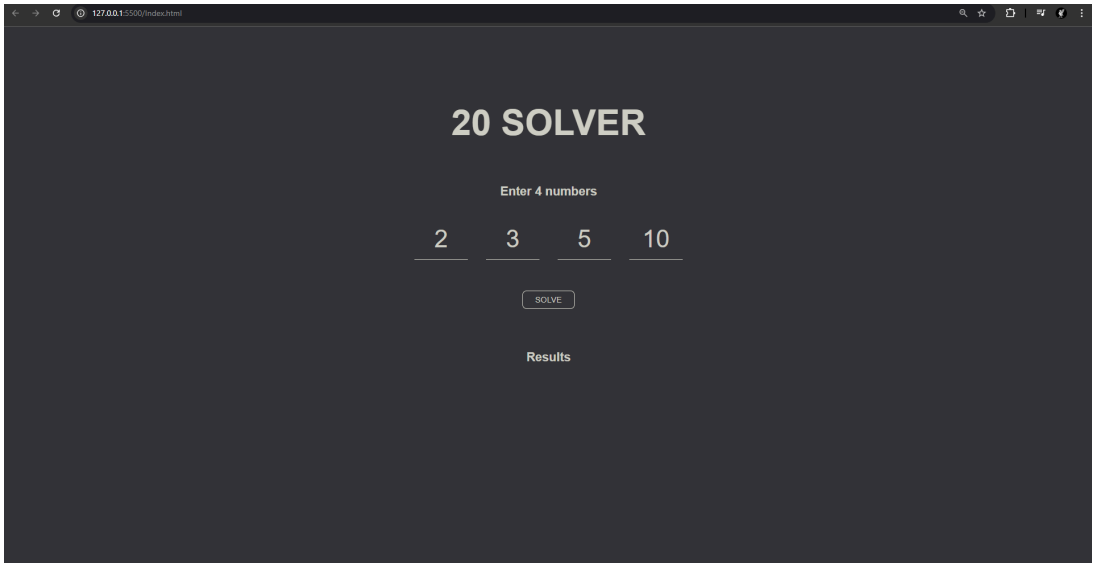
- Jika ekspresi memenuhi syarat (hasilnya 20 dan tidak ada pembagian dengan nol), ekspresi tersebut disimpan dalam Set yang bernama results.
- Set digunakan agar hanya ekspresi yang unik (tanpa duplikasi) yang disimpan.

D. Pengujian Program

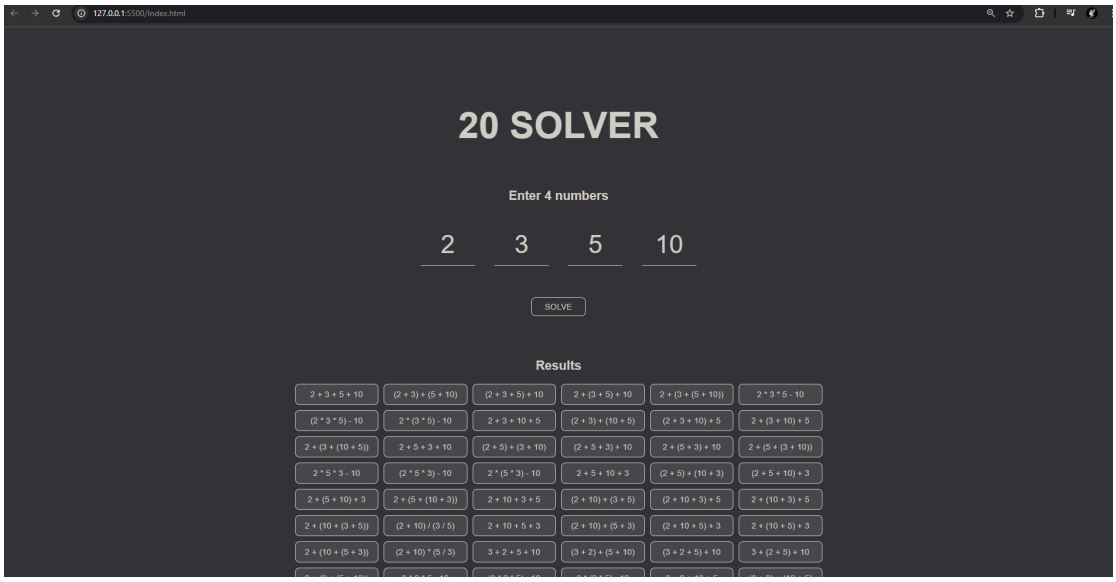
1. Halaman pertama ketika mengunjungi web, program akan meminta pengguna untuk menginput 4 angka



2. Setelah menginput 4 angka, pengguna menekan tombol solve

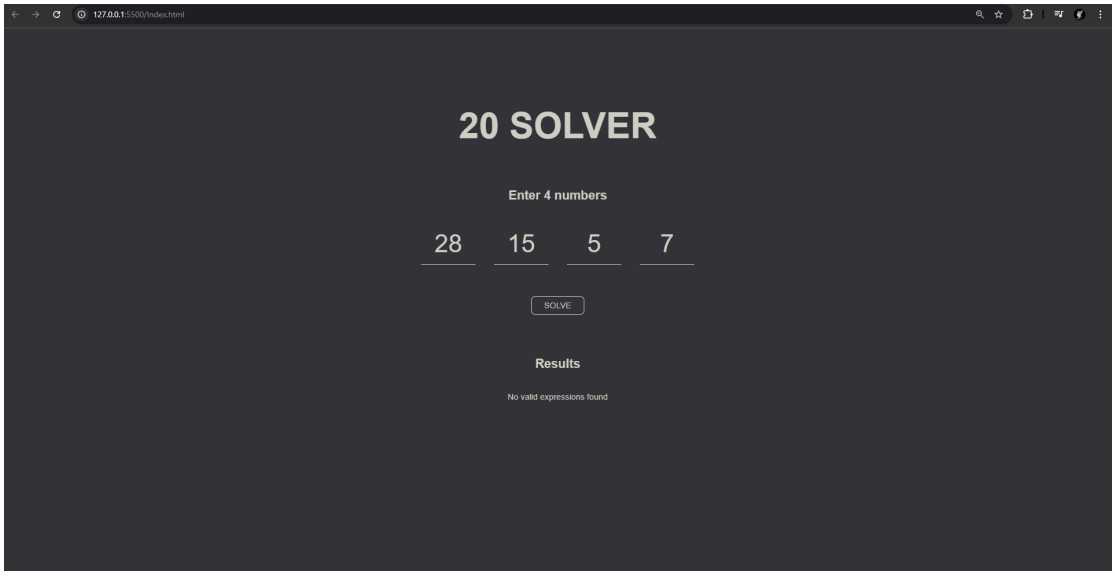


3. Nanti program akan generate semua ekspresi yang mungkin



4. Jika tidak ada ekspresi yang mungkin maka program akan mengeluarkan output:

   ‘No valid expressions found’



5. Jika pengguna tidak menginput angka yang valid maka program akan mengeluarkan output: “Please enter valid numbers.”