WASM vs JS

Grupo 8

Vicente Cruz

José Tomás Rebolledo

Vicente Akel

DEMO

Comparación de algoritmos

- JavaScript:

```
// Save the number of 2s that divide n
while (n % 2 == 0) {
    factors.push(2);
    n = Math.floor(n / 2);
}
```

- Código C:

```
int index = 0;
// Save the number of 2s that divide n
while (n % 2 == 0) {
    factors[index++] = 2;
    n = n / 2;
}
```

Código extraido de: https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/

Comparación de algoritmos

- JavaScript:

```javascript
for (let i = 3; i * i <= n; i = i + 2) {
    // While i divides n, push i and divide n
    while (n % i == 0) {
        factors.push(i);
        n = Math.floor(n / i);
    }
}
```

- Código C:

```c
for (unsigned long int  i = 3; i * i <= n; i = i + 2) {
    // While i divides n, store i and divide n
    while (n % i == 0) {
        factors[index++] = i;
        n = n / i;
    }
}
```

Código extraido de: https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/

Comparación de algoritmos

- JavaScript:

```
// This condition is to handle the
// case when n is a prime number
// greater than 2
if (n > 2) {
    factors[index++] = n;
}
```

- Código C:

```
// This condition is to handle the
// case when n is a prime number
// greater than 2
if (n > 2)
    factors.push(n);
```

Código extraido de: https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/

Optimización en WebAssembly

- O0
- O1
- O2
- **O3**
- Og
- Os
- Oz

O0

- **Descripción:** Sin optimizaciones. Recomendado para iniciar la portabilidad de un proyecto. Incluye varias comprobaciones para ayudar en el desarrollo.
- **Uso Típico:** Ideal para builds incrementales rápidos y para desarrollo inicial.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

O1

- **Descripción:** Optimizaciones simples. Reduce las comprobaciones de runtime en comparación con -O0, optimizando el código pero no al máximo.
- **Uso Típico:** Buen balance entre velocidad de compilación y optimización del código.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

O2

- **Descripción:** Habilita más optimizaciones que O1, incluyendo optimizaciones específicas de JavaScript.
- **Uso Típico:** Recomendado para builds de prueba donde el rendimiento es más importante pero sin llegar a la máxima optimización.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

O3

- **Descripción:** Optimizaciones adicionales que pueden requerir más tiempo de compilación. Optimiza el código al máximo.
- **Uso Típico:** Ideal para builds de lanzamiento donde el rendimiento es crítico.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

Og

- **Descripción:** Similar a O1, pero en futuras versiones, podría deshabilitar ciertas optimizaciones para mejorar la depurabilidad.
- **Uso Típico:** Para desarrollo cuando la depurabilidad es más importante que el rendimiento extremo.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

Os

- **Descripción:** Optimiza el código priorizando la reducción del tamaño del código sobre la velocidad de ejecución.
- **Uso Típico:** Útil cuando se necesita reducir el tamaño del binario final, especialmente en entornos donde la cantidad de datos a cargar es crítica.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

Oz

- **Descripción:** Extiende Os enfocándose aún más en minimizar el tamaño del código, lo cual puede tomar más tiempo de compilación.
- **Uso Típico:** Para cuando el tamaño del código es la prioridad absoluta, sacrificando potencialmente algo de rendimiento.

Fuente: https://emscripten.org/docs/tools_reference/emcc.html#emcc-o0

¡Muchas gracias!

AssembliRápido

VS

VelozScript