Errores Comunes

Por Ariel Parra & Cristian Donato

[ΓΩ5]

Tipos de Errores

En teória solo existen dos tipos de errores, errores de lógicos y errores de código, pero para una mejor distinción elegimos estos tipos de errores:

- Errores de Acceso a Memoria
- Errores de Precisión y Cálculo
- Errores de Lógica y Algoritmo
- Errores de Entrada/Salida
- Errores de Formato y Requerimientos

Teacher: What's so funny?
Me: Nothing
My brain:

Ram ramming ram in a Ram

Errores de Acceso a Memoria

1. **Acceso Fuera de los Límites de un Arreglo**:
    - Acceder a elementos fuera del rango declarado de un arreglo puede resultar en valores basura.

```
int arr[5] = {1, 2, 3, 4, 5};
int out_of_bounds = arr[10]; // Error: acceso fuera de los límites
```

2. **Variables No Inicializadas**:
    - Las variables no inicializadas pueden contener valores basura y causar comportamiento indefinido.

```
int x; // No inicializado
int y = x + 10; // Error: uso de variable no inicializada
```

3. **Fallo al Reiniciar Variables Globales**:

   - Las variables globales pueden retener valores entre casos de prueba, causando errores.

```
int global_var = 0;
void reset_global() {
    global_var = 0; // Reinicia la variable global entre casos
}
```

4. **Desbordamiento de la Pila con Recursión**:

   - La recursión profunda puede consumir demasiada memoria en la pila.

```
void recursive_function(int n) {
    if (n == 0) return;
    recursive_function(n - 1); // Puede causar desbordamiento de la pila si n es muy grande
}
```

5. **Múltiples Llamadas a una Función**:

   - Por ejemplo, llamar a la función `strlen()` o `.size()` en lugar de declarar una variable constante que albergue dicho valor.

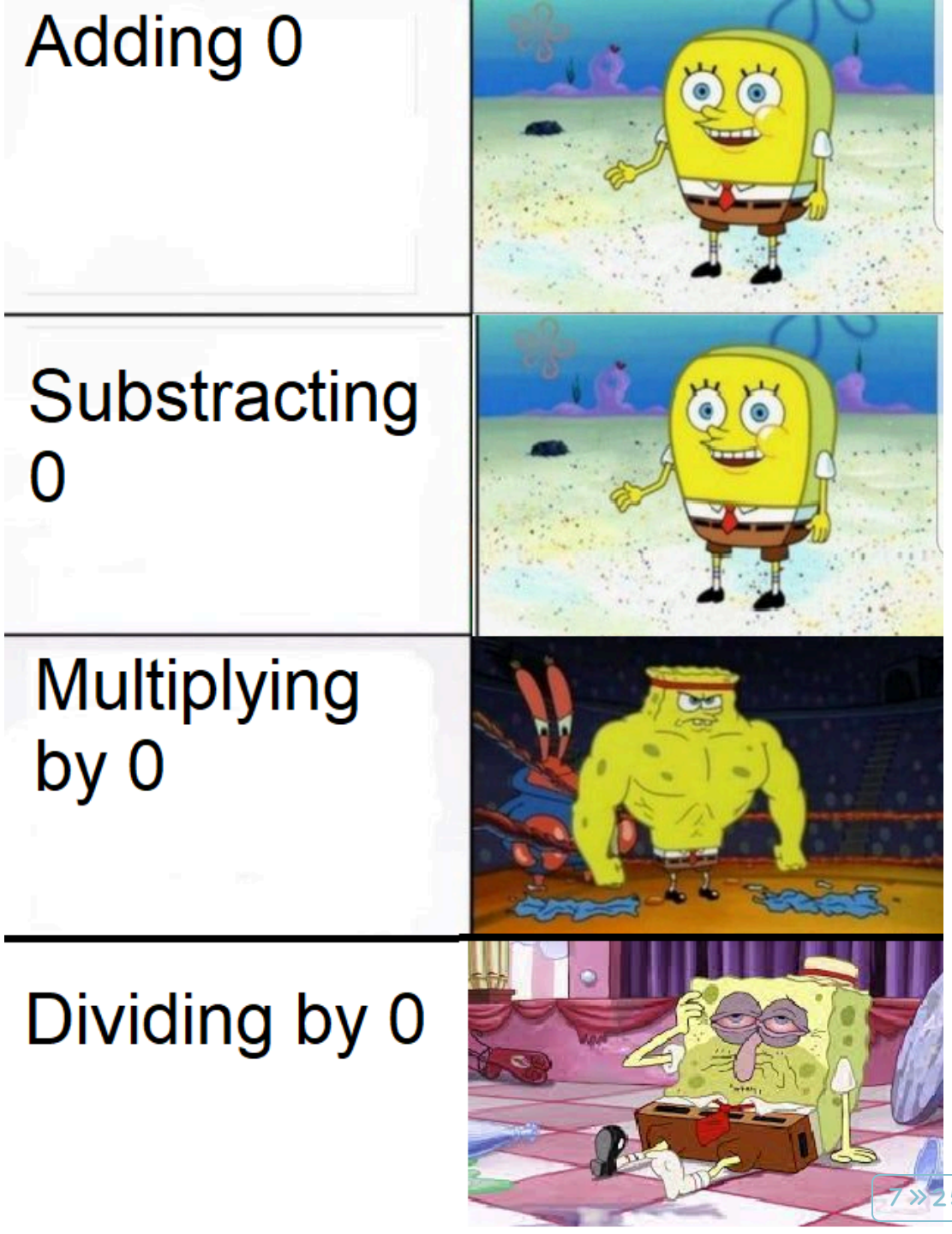
```
string str = "example";
size_t len = str.size(); // Mejor declarar una variable que llamar a str.size() múltiples veces
```

6. **Error al iterar en un vector**:

   - Recuerda que se recorrera desde `n-1` hasta `0`

```
for (int i = n; i >= 0; --i)
   cout << vec[i] << " ";// vec[n] no existe!, vec[n-1] si
for (int i = 0; i < n ; ++i)
   cout << vec[i+1] << " ";// vec[i+1] sobrepasa el limite!, limita a n-1
```

Errores de Precisión y Cálculo



1. **Errores de Precisión con Números de Punto Flotante**:
    - Los números de punto flotante pueden introducir errores de precisión, especialmente con números extremos.

```
float a = 1.0000001f;
float b = 1.0000002f;
if (a == b) {
    // Error: los números de punto flotante pueden no ser exactamente iguales
}
```

2. **Problemas con la Aritmética Modular**:
    - Manejar incorrectamente los números negativos con operaciones modulares puede llevar a resultados inesperados.

```
int a = -5 % 3; // Resultado puede ser inesperado
```

3. **Uso de `sqrt` y Conversión a Entero**:

   - La conversión incorrecta de valores `double` a enteros puede causar errores.

```
double x = 8.0;
int y = sqrt(x); // Puede perder precisión al convertir a int
```

4. **Overflow y Underflow de Enteros**:

   - Realizar operaciones que exceden el rango de un entero puede causar resultados incorrectos o comportamiento indefinido.

```
int a = INT_MAX;
int b = a + 1; // Overflow: comportamiento indefinido
```

5. **Underflow de Enteros Sin Signo**:

   - Restar de un entero sin signo puede llevar a desbordamientos inesperados.

```
unsigned int a = 0;
unsigned int b = a - 1; // Underflow: resultado inesperado
```

6. **Diferencias entre `floor` y `trunc` en Divisiones**:

   - Esto es importante al manejar números negativos: `floor(-4.5)` es `-5`, mientras que `trunc(-4.5)` es `-4`.

```
double a = -5.5;
int result1 = floor(a / 2); // -3
int result2 = trunc(a / 2); // -2
```

Errores de Lógica y Algoritmo

1. **Existencia de casos base**:
    - Considerar los casos 0, 1, y casos máximos para asegurar una solución optima.

```
void solve(int n) {
    if (n == 0) { /* manejar caso 0 */ }
    if (n == 1) { /* manejar caso 1 */ }
    if (n == INT_MAX) { /* manejar el caso máximo */ }
}
```

2. **Errores de Sintaxis**:
    - Errores básicos como olvidarse de un punto y coma o usar el tipo de datos incorrecto pueden causar fallos de compilación.

```
int x = 10 // Error de sintaxis: falta el punto y coma
```

3. **Sombreado de Variables (Variable Shadowing)**:

   - Declarar una variable en un ámbito interno con el mismo nombre que una externa puede causar comportamiento inesperado.

```
int x = 5;
void func() {
    int x = 10; // Sombrea la variable global x
}
```

4. **Orden de Operaciones (Precedencia de Operadores)**:

   - No entender el orden de las operaciones puede llevar a errores lógicos.

```
int x = 2 + 3 * 4; // El resultado es 14, no 20
```

5. **Errores en Comparadores Personalizados para Ordenamiento**:

   - Usar comparadores personalizados incorrectamente puede llevar a una ordenación errónea.

```
sort(vec.begin(), vec.end(), [](int a, int b) { return a > b; });
```

6. **En Caso de TLE, Checa que Todos los Ciclos Terminen**:

   - Asegúrate de que todos los ciclos en tu programa terminen para evitar tiempos de ejecución largos.

```
while (true) {
    // Error: bucle infinito que puede causar TLE
}
```

7. **Si una DP Multicaso Da TLE, Intenta Precalcular Todo**:

   - Precalcular datos para evitar recalcular en cada caso de prueba.

```
int dp[1000];
void precalculate() {
    for (int i = 0; i < 1000; ++i) {
        dp[i] = /* algún cálculo */;
    }
}
```

Errores de Entrada/Salida

1. **Manejo Incompleto de la Entrada**:
    - No leer toda la entrada puede llevar a procesar datos sobrantes de casos anteriores.

```
int x;
while (cin >> x) {
    // Procesar x, pero podría quedar entrada sin procesar
}
```

2. **No Usar `endl` (a Menos que Quieras Hacer `flush`)**:
    - Usa `\n` en lugar de `endl` para evitar flushing innecesario.

```
cout << "Hello\n"; // Mejor que usar cout << "Hello" << endl;
```

3. **Recordar Usar la Implementación de `cout` y `cin` Rápidos**:

   - Añadir configuraciones al inicio del `main` para optimizar el rendimiento de `cin` y `cout`.

```
ios::sync_with_stdio(0);
cin.tie(0);
```

4. **No Mezclar `cin`/`cout` con `scanf`/`printf`**:

   - Mezclar diferentes métodos de entrada/salida puede causar problemas de sincronización.

```
int x;
scanf("%d", &x);
cout << x; // Error: mezclar scanf con cout puede causar problemas
```

5. **Lectura de Espacios**:

   - Usar `getline` para leer cadenas con espacios.

```
string str;
getline(cin, str);
```

Errores de Formato y Requerimientos

1. **¿Alguna Variable Necesita Ser `long long` o `unsigned long long`?**:
    - Verifica si necesitas usar tipos más grandes como `long long` para evitar desbordamientos.

```
long long large_num = 1000000000000LL;
unsigned long long huge_num = 10000000000000000ULL;
```

2. **Si el Programa es Multicaso, Limpia las Variables**:
    - Asegúrate de reiniciar variables globales o estáticas entre casos de prueba.

```
int t; // Número de casos
while (t--) {
    // Reinicia variables aquí
    global_var = 0;
}
```

3. **¿El Formato de Salida es Fuera de lo Común?**:
    - Asegúrate de que

tu salida sigue el formato específico requerido.

```
cout << setfill('0') << setw(2) << hours << ":"<< setfill('0') << setw(2) << minutes ;
```

4. **¿El Límite es Más Chico de lo que Parece?**:
    - Revisa los límites del problema para evitar asumir un rango mayor.

```
int n = 1000; // Verifica que el límite es correcto
```

5. **¿El Problema Requiere de un Módulo en la Respuesta?**:

   - Asegúrate de aplicar el módulo a todos los cálculos relevantes.

```
int result = (a + b) % 1000000007; // Aplicar el módulo en la respuesta
```

6. **Si una Función Debe Devolver un Valor, ¿Estás Devolviéndolo en Todos los Casos?**:

   - Asegúrate de que todas las rutas de código devuelvan un valor si se espera.

```
int func(int x) {
    if (x > 0) return x;
    // Error: no se devuelve ningún valor si x <= 0
}
```

7. **Si el Problema Tiene Grafos, ¿Éstos Deben Ser Conexos?**:
    - Verifica la conectividad (de todos los nodos) si es un requisito del problema.

```
vector<int> graph[100];
bool visited[100] = {false};
void dfs(int node) {
    visited[node] = true;
    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor);
        }
    }
}   //
```

Los errores te hacen más fuerte, mi compa que puso `cout >>`:

Problemas

- **268A** Games ⮭
- **1919A** Wallet Exchange ⮭

Referencias

- Colin Galen. (2021). *C++ Mistakes Noobs Make (and how to prevent them)* [video]. Recuperado de https://www.youtube.com/watch?v=GsQM0nJhXws
- Meza, G. (2021). *Pendejario.txt*. Recuperado de https://github.com/GustavoMeza/icpc-notebook/blob/master/Pendejario.txt