Clase #09 de 27
Introducción a la Recursividad

Jun 18, Miércoles

Agenda para esta clase

- Revisión Trabajo #2
  - Casteo
  - osfreams
  - SVG
- Introducción a la Recursividad
- Factorial

Revisión Trabajo #2

Casteo

Streams

SVG

Revisión Trabajo #2

- Casteo
- Streams
- SVG

Casteo (!?)

Casteo (Casting)

- Reinterpreta el valor de una expresión como si fuese un valor de otro tipo de dato
- Clásico, menos claro e identificable
  - `(double)3`
  - `(int)3.14`
- Moderno, más claro e identificable
  - `static_cast<double>(3)`
  - `static_cast<int>(3.14)`
  - Otros casteos
    - `const_cast`
    - `dynamic_cast`
    - `reinterpret_cast`
- ¿Cuándo usar Casteo?
  - *NUNCA*
  - Bueno... *caaasi* nunca
  - Es un indicador de que hay que revisar el diseño o modelado de tipo, o que se está haciendo un procesamiento de bajo nivel; de cualquier manera, un casteo requiere verificación de su real necesidad o si hay que rediseñar parte de la solución

Streams No Standard

File Streams

Introducción a Streams No Standard

- El ya conocido `cout` es el flujo de salida *estándar*, y su tipo de dato es `ostream`
  - `cout << "Hello";`
  - Como es estándar, siempre está disponible

- También podemos crear flujos de salida *no estándar* que se conecten a archivos con el tipo de dato `std::ofstream` que está en el header `<fstream>`
  - `ofstream out{"output.txt"};`
  - `out << "Hello";`

```
<svg xmlns="http://www.w3.org/2000/svg">
   <text x="2" y="10">Introducción a SVG</text>
</svg>
```

Introducción a SVG

```svg
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" width="100" height="100">
    <polygon points="10,10 90,10 90,90" fill="blue"/>
    <rect x="25" y="25" height="50" width="50" fill="red"/>
    <circle cx="50" cy="50" r="20" fill="yellow" />
</svg>
```

- Es una notación, más precisamente un lenguaje formal, que sirve para describir forma textual gráficos en dos dimensiones (2D)
- Su sintaxis sigue la de XML, y por lo tanto la de HTML
- La notación también permite comentarios
  - `<!-- SVG es simple -->`

Cómo Emitir SVG desde un Programa C++

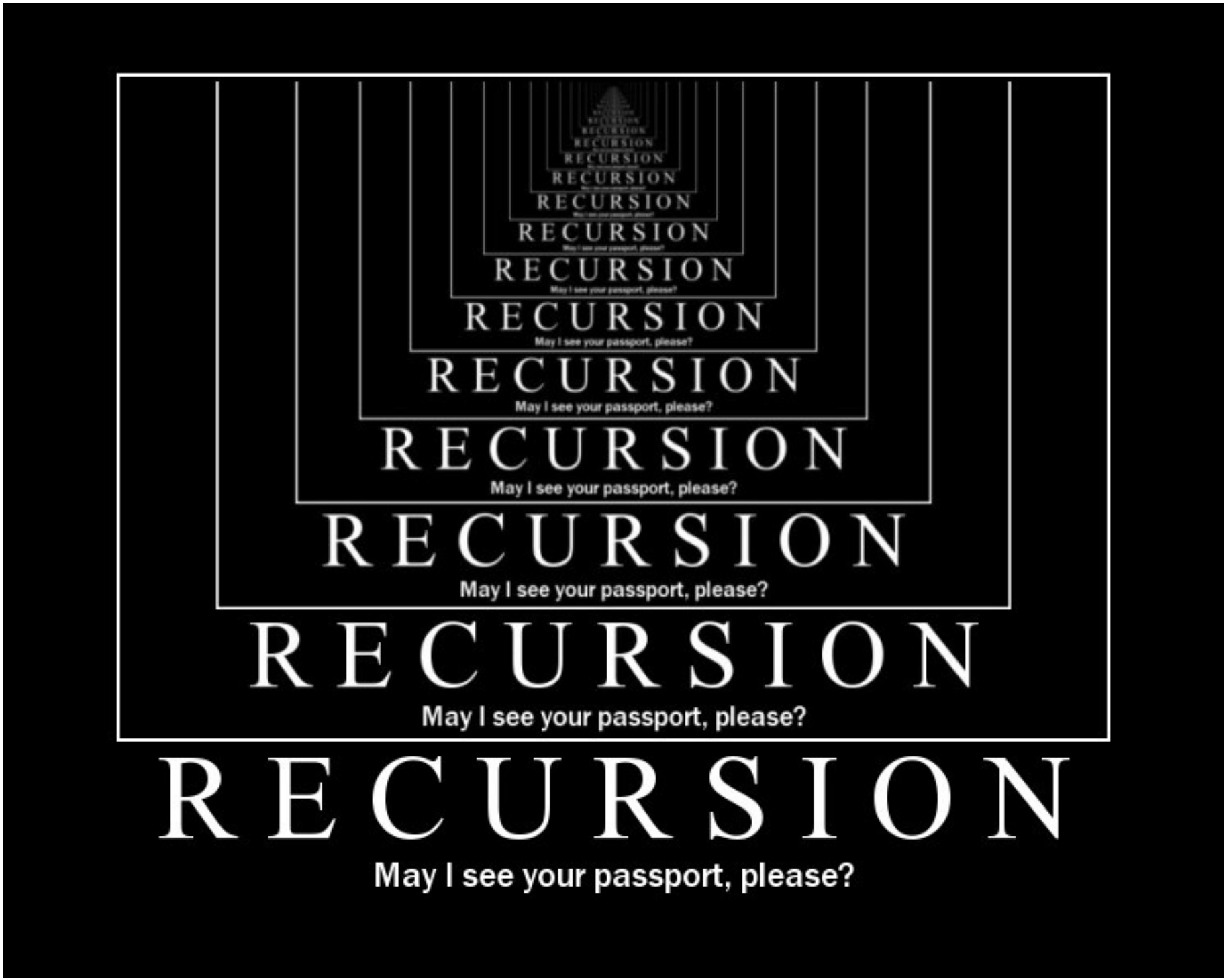
```cpp
#include<iostream>
#include<string>

using std::string;
using std::cout;

void EmitirSvg();
void AbrirSvg();
void CerrarSvg();
void DibujarFiguras();
void DibujarPolígono();
void DibujarRectángulo(string color);
void DibujarCírculo(unsigned radio, string color);

int main(){
    EmitirSvg();
}

void EmitirSvg(){
    AbrirSvg();
    DibujarFiguras();
    CerrarSvg();
}
```

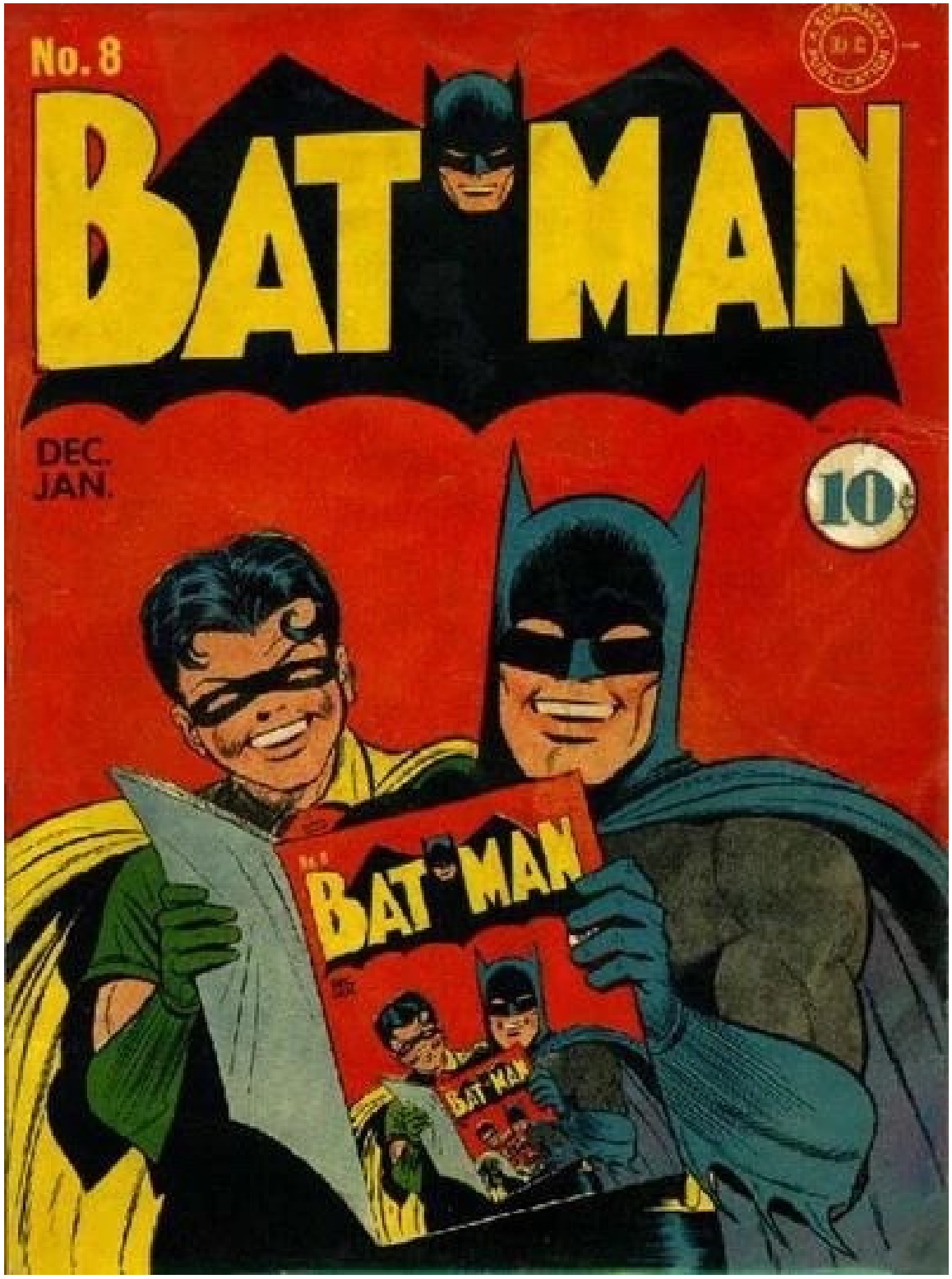
- `"\"Hola\"\n"s`
- `R"("Hola"\n)"s`

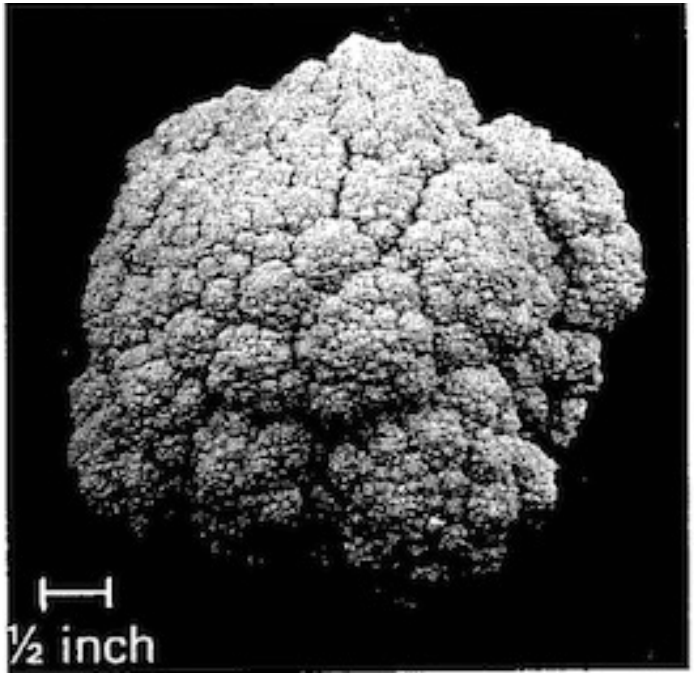
```cpp
void DibujarFiguras(){
    DibujarPolígono();            // Sin parámetros
    DibujarRectángulo("red");     // Ejemplo con parámetro color
    DibujarCírculo(20, "yellow"); // Con radio y color
}

void DibujarPolígono(){
    cout
        << R"(  <polygon points="10,10 90,10 90,90" fill="blue"/>)"
        << "\n";
}

void DibujarRectángulo(string color){
    cout
        << R"(  <rect x="25" y="25" height="50" width="50" )"
        << R"(fill=")" << color << R"(" />)"
        << "\n"
        ;
}
void DibujarCírculo(unsigned radio, string color){
    cout
        << R"(  <circle cx="50" cy="50" )"
        << R"(r=")" << radio << R"(" )"
        << R"(fill=")" << color << R"(" />)"
        << "\n"
        ;
}
```

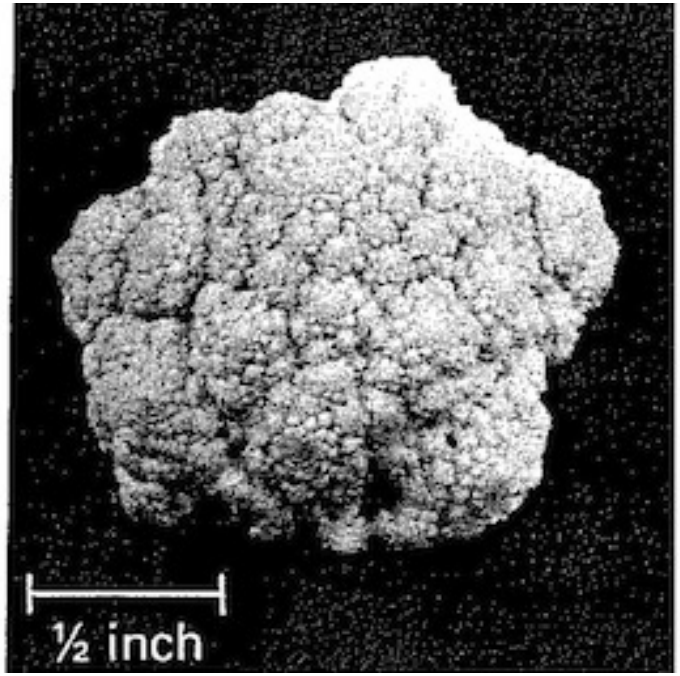
```cpp
void AbrirSvg(){
    cout << R"(<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="100" height="100">)" << "\n";
}

void CerrarSvg(){
    cout << R"(</svg>)" << "\n";
}
```

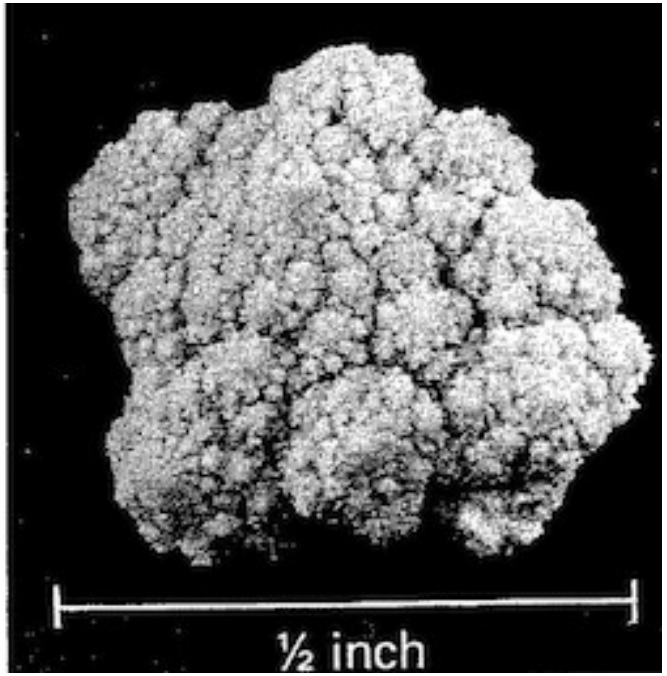
Introducción a la Recursividad


RECURSION
May I see your passport, please?


No. 8
BAT MAN
DEC.
JAN.
10¢
BAT MAN
BAT MAN

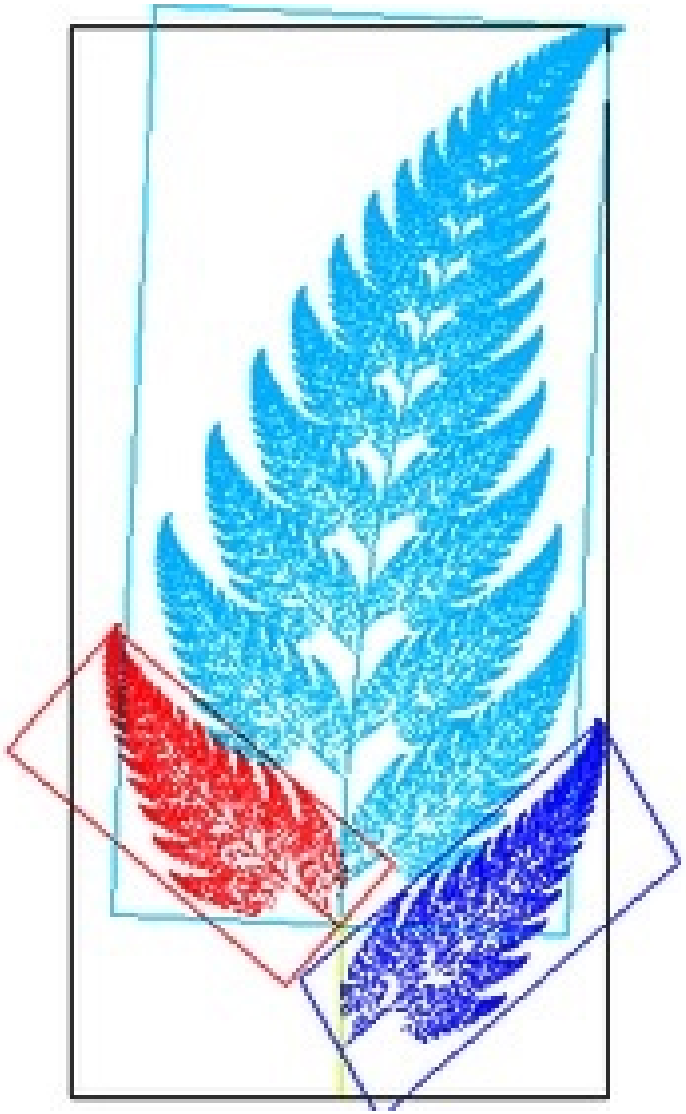

½ inch


½ inch


½ inch

¿Qué es la Recursividad?

- Es la definición de un proceso basado en su propia definición
- Es la repetición de ítems de una manera *autosimilar*
- "*Para entender recursividad, uno debe entender recursividad*"
- GNU
- Dimensión Temporal o de Proceso
  - Alternativa a la iteración
  - Definiciones de funciones de manera recursiva
- Dimensión Espacial o Estructural
  - Estructuras recursivas



Funciones Recursivas: Factorial

Ejemplo – Factorial: Definición

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times (n-1) \times n$$

$$n! = \begin{cases} 1 & n = 0 \\ (n-1)! \times n & n > 0 \end{cases}$$

$$\text{Factorial}: \mathbb{N} \to \mathbb{N} / \text{Factorial}(n) = \begin{cases} 1 & 0 \\ n \cdot \text{Factorial}(n-1) & \text{e.o.c.} \end{cases}$$

$$\text{Factorial}: \mathbb{N} \to \mathbb{N} / \text{Factorial}(n) = \begin{cases} 1 & n < 2 \\ n \cdot \text{Factorial}(n-1) & \text{e.o.c.} \end{cases}$$

Ejemplo – Factorial: Evaluación

$$\text{Factorial}: \mathbb{N} \to \mathbb{N} / \text{Factorial}(n) = \begin{cases} 1 & n < 2 \\ n \cdot \text{Factorial}(n-1) & \text{e.o.c.} \end{cases}$$

$$\begin{aligned}
&\text{Factorial}(4) && = \\
&4 \cdot \text{Factorial}(4-1) && = \\
&4 \cdot 3 \cdot \text{Factorial}(3-1) && = \\
&4 \cdot 3 \cdot 2 \cdot \text{Factorial}(2-1) && = \\
&4 \cdot 3 \cdot 2 \cdot \text{Factorial}(1) && = \\
&4 \cdot 3 \cdot 2 \cdot 1 && = \\
&4 \cdot 3 \cdot 2 && = \\
&4 \cdot 6 && = \\
&24
\end{aligned}$$

Ejemplo – Factorial: Pruebas e Implementación

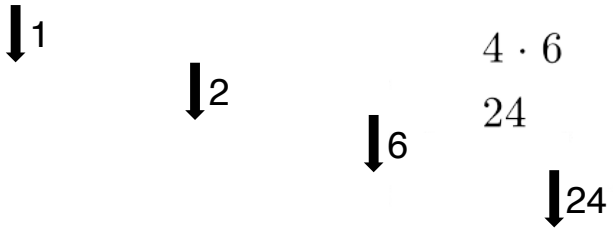
$$\text{Factorial}: \mathbb{N} \to \mathbb{N} / \text{Factorial}(n) = \begin{cases} 1 & n < 2 \\ n \cdot \text{Factorial}(n-1) & \text{e.o.c.} \end{cases}$$

```
unsigned long Fact(unsigned n){ return
    n < 2 ?             1:
            n * Fact(n-1);
}
```

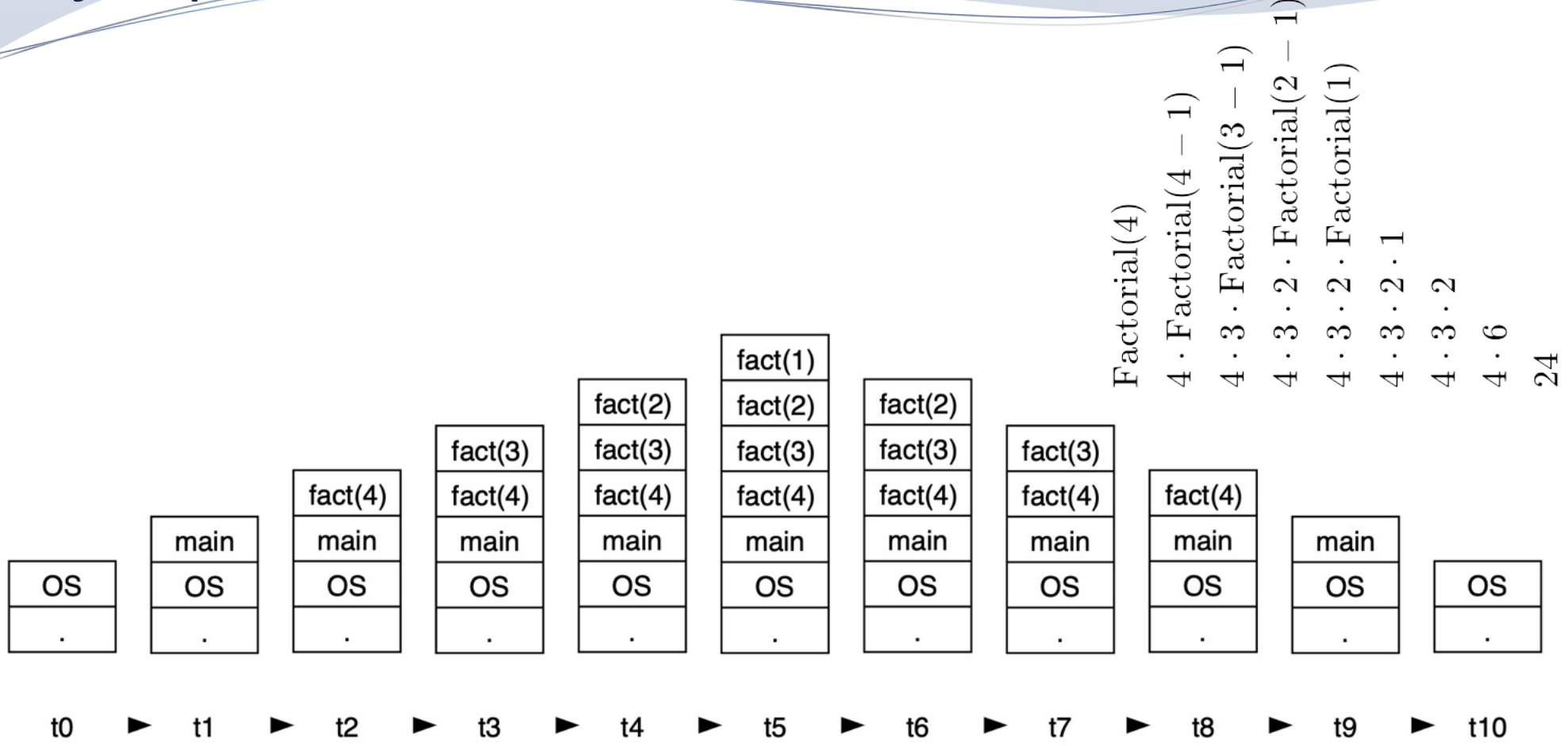
```
#include <cassert>

unsigned long Fact(unsigned);

int main(){
   assert(                    1 == Fact( 0) );
   assert(                    1 == Fact( 1) );
   assert(                    2 == Fact( 2) );
   assert(                    6 == Fact( 3) );
   assert(                   24 == Fact( 4) );
   assert(                  120 == Fact( 5) );
   assert(                  720 == Fact( 6) );
   assert(                 5040 == Fact( 7) );
   assert(                40320 == Fact( 8) );
   assert(               362880 == Fact( 9) );
   assert(              3628800 == Fact(10) );
   assert(             39916800 == Fact(11) );
   assert(            479001600 == Fact(12) );
   assert(           6227020800 == Fact(13) );
   assert(          87178291200 == Fact(14) );
   assert(        1307674368000 == Fact(15) );
   assert(       20922789888000 == Fact(16) );
   assert(      355687428096000 == Fact(17) );
   assert(     6402373705728000 == Fact(18) );
   assert(  121645100408832000 == Fact(19) );
   assert( 2432902008176640000 == Fact(20) );
}
```

Ejemplo – Factorial: Pila de Invocaciones (I)

$$\text{Factorial}(4)$$
$$4 \cdot \text{Factorial}(4-1)$$
$$4 \cdot 3 \cdot \text{Factorial}(3-1)$$
$$4 \cdot 3 \cdot 2 \cdot \text{Factorial}(2-1)$$
$$4 \cdot 3 \cdot 2 \cdot \text{Factorial}(1)$$
$$4 \cdot 3 \cdot 2 \cdot 1$$
$$4 \cdot 3 \cdot 2$$
$$4 \cdot 6$$
$$24$$



Ejemplo – Factorial: Pila de Invocaciones



Términos de la clase #09

Definir cada término con la bibliografía

- Casteo
- ofstream
- SVG
- Raw strings
- Introducción a la Recursividad (Recursión)
- Recursividad:
  - Invocación recursiva
  - Caso base
  - Definición de Factorial
  - Implementación recursiva de Factorial
  - Traza de invocaciones
  - Pila de invocaciones

¿Consultas?

Fin de la clase