Fundamentos de Programación

Unidad 8: Funciones

Pablo Novara

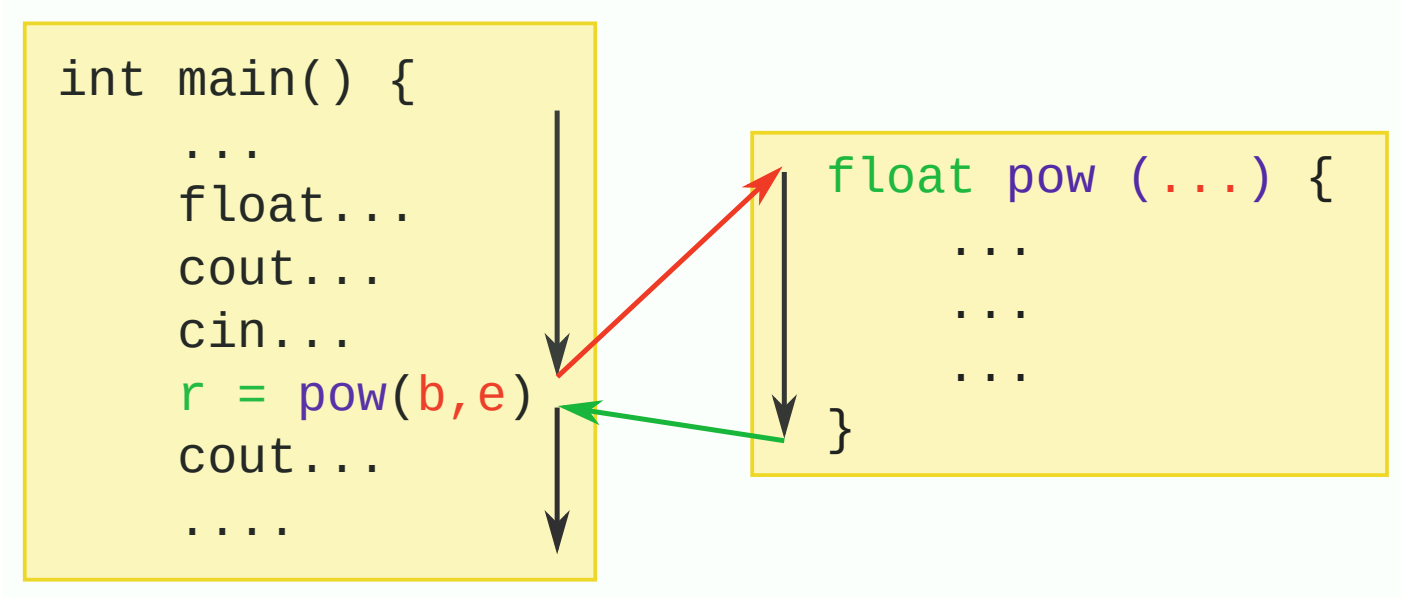
EJEMPLO

Escriba un programa que permita ingresar una base y un exponente, y muestre el resultado de la potenciación:

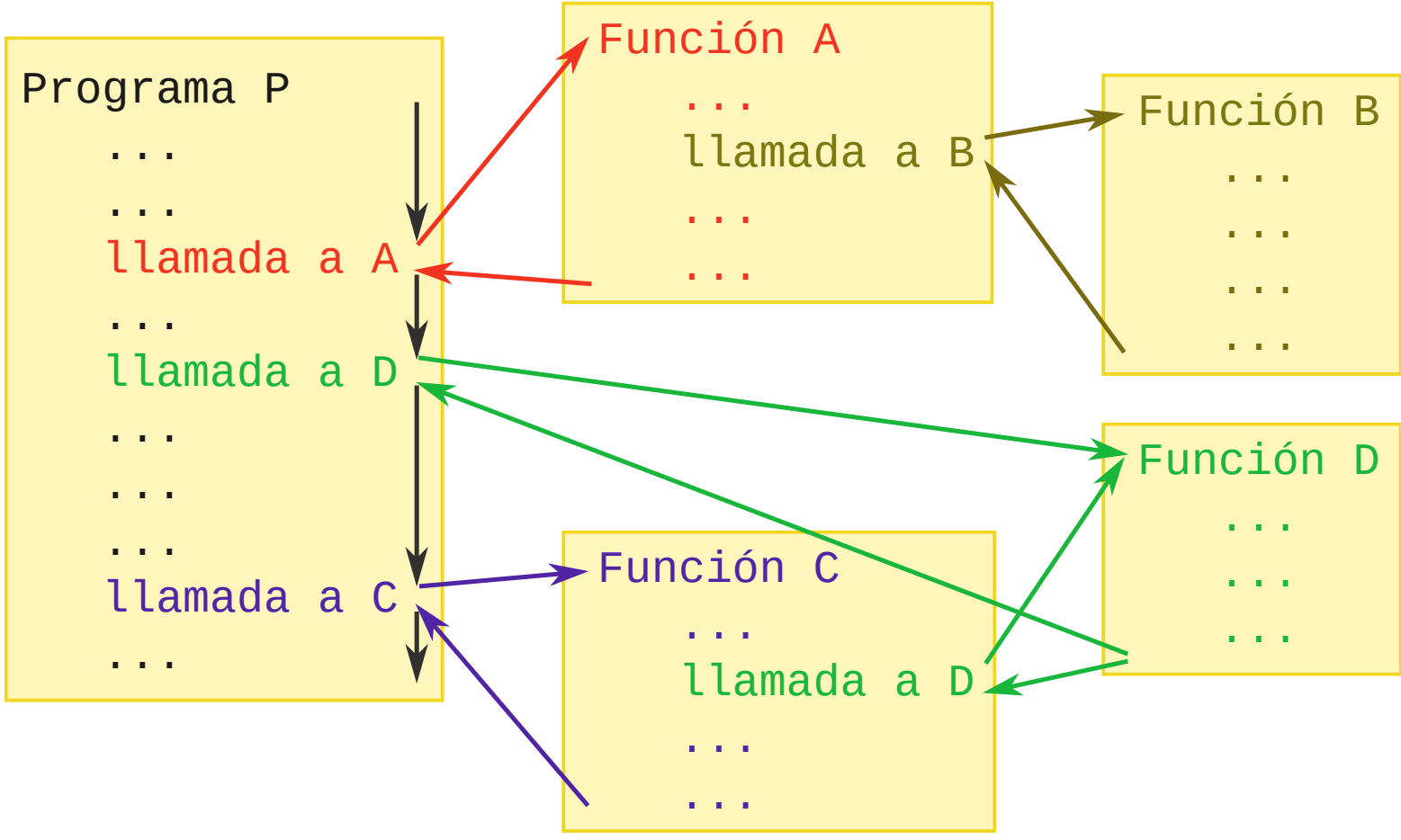
```
int main() {

    float base, exponente;
    cout << "Ingrese base y exponente: ";
    cin >> base >> exponente;

    float result = pow(base,exponente);

    cout << base << "^" << exponente
         << "=" << result << endl;

}
```

FUNCIONES

¿Qué ocurre al invocar una función?



FUNCIONES


Programa P
...
...
llamada a A
...
llamada a D
...
...
...
llamada a C
...
Función A
...
llamada a B
...
...
Función B
...
...
...
...
Función C
...
llamada a D
...
...
Función D
...
...
...

UTILIZACIÓN DE FUNCIONES

Forward Declaration:

```
int potencia(int base, int exponente);
```

*argumentos **formales***

Llamada:

```
int main() {
    int b,e;
    cin >> b >> e;
    int x = potencia(b, e);
    cout << x;
}
```

*argumentos **actuales***

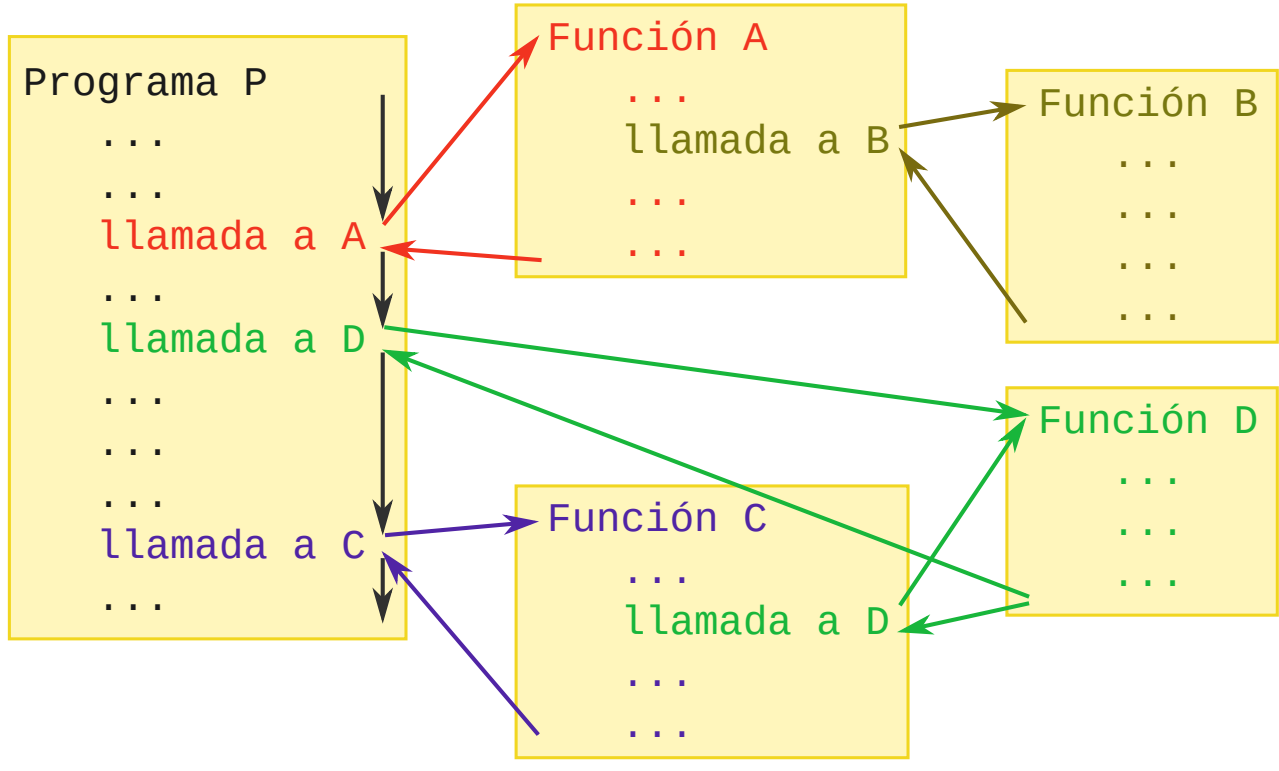
*¿Qué necesitamos conocer de una función para poder **utilizarla**?*

IMPLEMENTACIÓN DE FUNCIONES

```
tipo nombre ( argumentos ) {
    ...instrucciones...
    return valor_de_retorno;
}
```

```
int potencia ( int base, int exponente ) {
    int res = 1;
    for (int i=0; i<exponente; i++)
        res *= base;
    return res;
}
```

¿Qué necesitamos conocer de una función para poder **implementarla**?



> *The secret to building large apps is never build large apps. Break your applications into small pieces. Then, assemble those testable, bite-sized pieces into your big application.*

Justin Meyer

DEFINICIÓN VS DECLARACIÓN

Declaración (prototipo/caja negra):
indica cómo se usa (qué recibe y qué devuelve):

```
int potencia (int base, int exponente);
```

Definición (implementación/caja blanca):

```
int potencia (int base, int exponente) {
    int res = 1;
    for (int i=0;i<exponente;i++)
        res *= base;
    return res;
}
```

EJEMPLOS

1. Escriba un programa que utilizando un función calcule y muestre **el área de un círculo**.

```
void AreaCirculo() {
    float radio;
    cin >> radio;
    float area = M_PI*radio*radio;
    cout << "El area es: " << area;
}
```

MAL!!!

```
int main() {
    AreaCirculo();
}
```

MAL!!!

OPERACIONES DE ENTRADA Y SALIDA

```
void AreaCirculo() {
    float radio;
    cin >> radio;
    float area = M_PI*radio*radio;
    cout << "El area es: " << area;
}
```

MAL!!!

⊗ *No utilizar operaciones de entrada/salida en funciones*

```
float AreaCirculo(float radio) {
    float area = M_PI*radio*radio;
    return area;
}
```

Bien

OPERACIONES DE ENTRADA Y SALIDA

1. Escriba un programa que utilizando un función calcule y muestre **el área de un círculo**.

```
float AreaCirculo(float radio) {
    float area = M_PI*radio*radio;
    return area;
}
```

Bien

```
int main() {
    float radio;
    cin >> radio;
    float area = AreaCirculo(radio);
    cout << "El area es: " << area;
}
```

OPERACIONES DE ENTRADA Y SALIDA

1. Escriba un programa que utilizando un función calcule y muestre **el área de un círculo**.

```
float AreaCirculo(float radio);
```

2. Escriba un programa que calcule y muestre **el volumen de un cilindro**.

```
int main() {
    float radio, altura;
    cin >> radio >> altura;
    float vol = AreaCirculo(radio)*altura;
    cout << "El volumen es: " << vol;
}
```

EJEMPLOS

3. Escriba una función para determinar si un número es primo.
   - Un número N es primo si **solamente** es divisible por 1 y por N
   - **solamente** -> entonces tengo que demostar que no hay otro divisor
   - Conclusión: tengo que **buscar** divisores entre 2 y N-1

EJEMPLOS

3. Escriba una función para determinar si un número es primo.

```
bool es_primo(int n) {
    for(int i=2;i<n;i++) {
        // si encuentro un divisor,
        // ya con uno se que no es primo
        if (n%i==0) return false;
        // pero si no encuentro, no se
        // nada, hay que seguir buscando
    }
    // si ya probe todos y ninguno es
    // divisor, entonse sí era primo
    return true;
}
```

EJEMPLOS

3. Escriba una función para determinar si un número es primo.

4. Escriba un programa para encontrar los 100 primeros números primos.

```
int main() {
    int num_a_probar=2, cant_primos=0;
    while (cant_primos<100) {
        if (es_primo(num_a_probar)) {
            cout<<num_a_probar<<endl;
            cant_primos++;
        }
        num_a_probar++;
    }
}
```

VENTAJAS DEL USO DE FUNCIONES

- **Permiten reducir la complejidad**
- Logran mayor modularidad
- Facilitan el desarrollo en equipo
- Facilitan la prueba y depuración
- Optimizan el uso de memoria
- Evitan el copy/paste
- Permiten crear bibliotecas para reutilizar

PASAJE DE PARÁMETROS

Por Valor/Copia

El argumento contiene una *copia* del *valor* de la variable/expresión de la llamada.

```
void por_valor ( int x );
```

- No se modifican los parámetros actuales.
- Los argumentos pueden ser variables, constantes y/o expresiones.
- Puede haber un casteo implícito.

PASAJE DE PARÁMETROS

Por Referencia

El argumento es un alias de la variable de la llamada.

```
void por_referencia ( int &x );
```

- Se pueden modificar los parámetros actuales.
- Si no es const, sólo se puede utilizar una variable.
- El tipo debe coincidir exactamente.

PASAJE POR VALOR VS POR REFERENCIA

```
void por_valor(int  x) { x=42; } // copia
void por_refer(int &x) { x=42; } // alias

int main() {
    int p = 3;
    cout << p << endl; // muestra 3

    por_valor(p);
    cout << p << endl; // muestra 3

    por_refer(p);
    cout << p << endl; // muestra 42
}
```

¿CUANDO UTILIZAR PASAJE POR REFERENCIA?

- cuando es caro hacer copias:

```
void muestra(const matrix_1000x1000 &m)
```

*Si es para evitar la copia, agregar el **const***

- se requiere modificar las variables:

```
void swap( int &a, int &b );
```

- ~~se requiere devolver más de un resultado~~

```
void raices(float a, float b, float c,
            float &r1, float &r2);
```

Ya no se recomienda usar pasaje por referencia para retornar multiples valores

MÚLTIPLES VALORES DE RETORNO

Ejemplo: resolvente para raices reales:

```
tuple<float,float> raices (float a,
                           float b,
                           float c);
```

Llamada:

```
...
float raiz1, raiz2;
tie(raiz1,raiz2) = raices( a, b, c );
cout << raiz1 << endl
     << raiz2 << endl;
...
```

MÚLTIPLES VALORES DE RETORNO

Implementación:

```
tuple<float,float> raices (float a,
                           float b,
                           float c);
{
    float sqrt_d = sqrt(b*b-4*a*c);
    float r1 = (-b+sqrt_d)/(2*a);
    float r2 = (-b-sqrt_d)/(2*a);
    return make_tuple(r1,r2);
}
```

EJEMPLOS

5. Escriba un programa para obtener las raices de una ecuación cuadrática (sean reales o complejas).

Empezar por hacer el análisis y algunos ejemplos (a=-1,b=2,c=3 y a=1,b=2,c=5)

¿Por dónde empezar a "codificar"? ¿Funciones o programa cliente?

EJEMPLOS

```
bool tiene_raices_reales(float a, float b, float c);
tuple<float,float> calc_r_reales(float a, float b, float c);
tuple<float,float> calc_r_complejas(float a, float b, float c);

int main() {
    float a, b, c;
    cin >> >> b >> c;

    if (tiene_raices_reales(a,b,c)) {

        float raiz1, raiz2;
        tie(raiz1,raiz2) = calc_r_reales(a,b,c);
        cout << "x1 = " << raiz1 << endl;
        cout << "x2 = " << raiz2 << endl;

    } else {

        float preal, pimag;
        tie(preal,pimag) = calc_r_complejas(a,b,c);
        cout << "x1 = " << preal << " + " << pimag << "i" << endl;
        cout << "x2 = " << preal << " - " << pimag << "i" << endl;

    }
}
```

✅ *el main se encarga solo de la entrada/salida, y las funciones solo de la matemática necesaria*

PRINCIPIOS DE DISEÑO

- **Single Responsability Principle**:

  Cada función debe tener una y solo una responsabilidad.

- **Single Level of Abstraction**:

  En cada función, se deben observar detalles de un mismo nivel de abstracción.

PARÁMETROS POR DEFECTO

Argumentos por defecto:

```
int potencia(int base, int exp=2);
```

Llamada:

```
    ...
    int xe = potencia(x,e); // x^e
    ...
    int x2 = potencia(x);   // x^2
    ...
```

- Siempre al final, y solo en la declaración

SOBRECARGA DE FUNCIONES

Varias funciones con el mismo nombre:

```
float promedio (int a, int b);
float promedio (int a, int b, int c);
```

- Se distinguen por la **cantidad** de argumentos.

```
void swap (int &a,   int &b  );
void swap (float &a, float &b);
```

- Se distinguen los **tipos** de argumentos.

```
int   div(int a, int b);
float div(int a, int b);
```

Error

- **No** se distinguen por el tipo de retorno.

RECURSIVIDAD

6. ¿Qué hace y cómo funciona el siguiente programa?

```
void foo(int n) {
    if (n==0) {
        cout << "KBOOM!!!" << endl;
    } else {
        cout << n;
        cin.get(); // esperar un enter
        foo(n-1);
    }
}

int main() {
    foo(5);
}
```

¿Qué pasa si saco el `if` y dejo solo el cont. del `else`?

RECURSIVIDAD

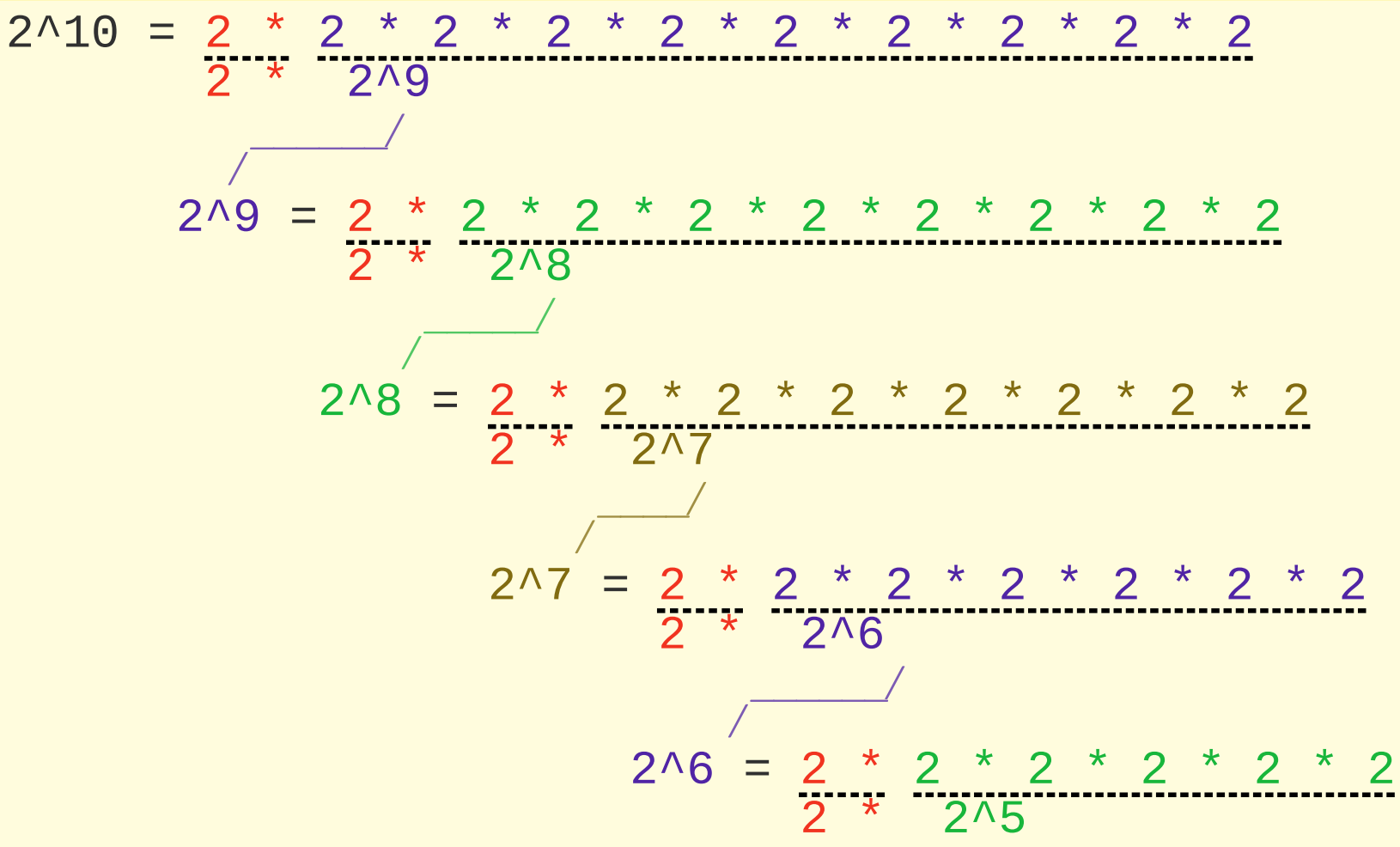
- Condición para una función sea recursiva:
  - Que se llame a sí misma.

- Condición de parada:
  - Debe llegar a algún caso en donde se resuelva sin recursividad.

⊗ *Si no hay condición de parada el algoritmo es "infinito"*

Ejemplos: potencia, factorial, fibonacci, ...

EJEMPLO: POTENCIA RECURSIVA



Regla de recursión: B^E = B * B^(E-1)

EJEMPLO: POTENCIA RECURSIVA

$$2^{10} = \underbrace{2 * \underbrace{2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2}_{2^9}}_{2 *}$$

$$2^9 = \underbrace{2 * \underbrace{2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2}_{2^8}}_{2 *}$$

$$2^8 = \underbrace{2 * \underbrace{2 * 2 * 2 * 2 * 2 * 2 * 2 * 2}_{2^7}}_{2 *}$$

$$2^7 = \underbrace{2 * \underbrace{2 * 2 * 2 * 2 * 2 * 2 * 2}_{2^6}}_{2 *}$$

$$2^6 = \underbrace{2 * \underbrace{2 * 2 * 2 * 2 * 2 * 2}_{2^5}}_{2 *}$$

... => **2^0 = 1**

Criterio de corte/parada: B^0 = 1

EJEMPLO: ITERATIVA VS. RECURSIVA

```
int potencia(int b, int e) {
    int r = 1;
    for( int i = 0; i<e; ++i )
        r *= b;
    return b;
}
```

```
int potencia(int b, int e) {
    if ( e==0 )
        return 1;
    else
        return b * potencia(b, e-1);
}
```

¿cual es mejor?

EJEMPLO: POTENCIA RECURSIVA

$$2^{16} = \frac{2*2*2*2*2*2*2*2}{2^8} * \frac{2*2*2*2*2*2*2*2}{2^8}$$

$$2^8 = \frac{2*2*2*2}{2^4} * \frac{2*2*2*2}{2^4}$$

$$2^4 = \frac{2*2}{2^2} * \frac{2*2}{2^2}$$

$$2^2 = 2^1 * 2^1$$

Nueva regla de recursión: $B^E = B^{(E/2)} * B^{(E/2)}$

solo para exponentes pares

EJEMPLO: ITERATIVA VS. RECURSIVA

```
int potencia(int b, int e) {

    if ( e==0 ) {
        return 1;
    } else  {

        if (e%2==0) {
            int x = potencia(b, e/2);
            return x * x;

        } else {
            return b * potencia(b, e-1);

        }

    }
}
```