

UNT – FACEyT - DEEC



Programación I

Año 2016

Unidad 2: Funciones en Lenguaje C - Introducción

- George Polya (1887-1985) en el libro “Cómo plantear y resolver problemas” propone desglosar los problemas en pequeños problemas con el fin de encontrar la solución.
- Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa, reduciendo la cantidad de código del programa en sí, reutilizando código. Aplicación del paradigma “Dividir y conquistar”.

Unidad 2: Funciones en Lenguaje C - Introducción

- Las funciones, en lenguaje C, están constituidas por un conjunto de instrucciones (bloque de código) que realizan una tarea específica.
- A las funciones se las identifican con un nombre.
- En C, las funciones deben declararse (declaración), implementarse (definición) y usarse (invocar o llamar).
- Cuando una función es "*llamada*" o "*invocada*", ésta toma el control del programa hasta que finalice de ejecutar su bloque de código. Una vez que la función finalizó, el programa sigue ejecutándose en el punto en donde se "*invocó*" a la función.

Unidad 2: Funciones en Lenguaje C - Introducción

- Si la función "*llamadora*" necesita enviar valores a la función "*llamada*" o "*invocada*" lo hace a través de argumentos, que se reciben por medio de parámetros.
- Puesto que la transferencia de datos es de argumentos a parámetros, estos deben coincidir en cantidad, posición y tipo de datos.
- Una función puede no tener parámetros, por lo tanto la "*llamada*" tampoco tiene argumentos.
- La lista de argumentos, en la llamada de la función, se indican, entre paréntesis, a continuación del nombre de la función que se "*llama*" o "*invoca*".
- La lista de parámetros se indica en la "*cabecera*" de la definición de la función, precedidos del tipo de dato.

Unidad 2: Funciones en Lenguaje C - Introducción

- Las funciones en C devuelven un sólo valor o ninguno.
- Las funciones que no devuelven valor se denominan funciones **void**.
- Para llamar o invocar un función se debe tener en cuenta si es **void** o devuelve valor.
- Si la función es **void** (no devuelve valor) se la "llama" o "invoca" una sentencia simple:

***nombre* ([lista de argumentos]);**

Unidad 2: Funciones en Lenguaje C - Introducción

Si no es **void**, es decir "*devuelve valor*", la sintaxis de llamada pueden ser:

1.- En una estructura secuencial que involucre una asignación:

variable = *nombre* ([lista de argumentos]);

el tipo de dato de la variable debe ser del mismo tipo de dato que devuelve la función.

2.- En una sentencia de salida de datos:

printf(" ...", *nombre* ([lista de argumentos]));

3.- En una sentencia simple si no se utiliza el valor de retorno:

***nombre* ([lista de argumentos]);**

Unidad 2: Funciones en Lenguaje C - Introducción

- En C las funciones pueden ser:
 - estándar o
 - definidas por el usuario.
- Las funciones estándar están definidas de acuerdo a la estandarización ANSI-ISO C-99.
- La funciones definidas por el usuario son aquellas que el programador define de acuerdo a sus necesidades.

Unidad 2: Funciones estándar o de bibliotecas

- Estas funciones ya están definidas en las bibliotecas estándar del Lenguaje C.
- El lenguaje C tiene 15 bibliotecas, las más usuales son:
 - **ctype.h**: funciones de caracteres.
 - **math.h**: funciones matemáticas.
 - **stdio.h**: funciones de entrada y salida estándar.
 - **stdlib.h**: funciones especiales.
 - **string.h**: funciones de cadenas de caracteres.
 - **time.h**: funciones de tiempo, fecha y hora.

Unidad 2: Funciones estándar o de bibliotecas

- Para utilizar las funciones de las bibliotecas estándar debe **incluirse** en la compilación los "*archivos cabecera*" o "*headers*" que contienen la librería correspondiente. Esto debe hacerse en la cabecera del programa con la palabra reservada **include**
- La sintaxis para incluir una librería en la compilación es:
 1. Si el archivo cabecera está en el directorio por defecto:
#include <nombre.h>
 2. Si el archivo cabecera NO está en el directorio por defecto:
#include "[path]nombre.h"
 3. Si el archivo cabecera está en el directorio de trabajo:
#include "nombre.h"

Unidad 2: Funciones definidas por el usuario

Las funciones definidas por el usuario deben ser:

1. declaradas (declaración),
2. implementadas (definición) y
3. usadas (invocar o llamar).

1. Declaración: Se declaran en la zona de declaraciones globales, antes de la función principal (***int main(...)***).

Sintaxis de la declaración:

tipo_de_dato nombre ([lista de parámetros]);

La declaración se realiza a los fines de compatibilizar identificadores y tipos de datos durante la compilación.

Unidad 2: Funciones definidas por el usuario

- **tipo_de_dato:** Es el tipo de dato que devuelve la función. Puede devolver cualquier tipo de dato estándar de C o definido por el usuario, **EXCEPTO arreglos**. Si la función no devuelve ningún valor se usa la palabra reservada **void**. Por defecto devuelve un entero.
- **nombre:** debe ser un identificador válido de C. Con este nombre se llamará o invocará a la función.
- **lista de parámetros:** es una lista de nombres, cada uno precedido por el tipo de dato y separados por coma. Esta lista es una "declaración implícita" de variables locales de la función. Si la función no tiene parámetros lo mismo deben ponerse los paréntesis.

Unidad 2: Funciones definidas por el usuario

2. Definición: Se definen luego del cierre de la llave de bloque de código de la función principal (***int main(...)***).

Sintaxis:

//cabecera de la función

tipo_de_dato nombre ([lista de parámetros])

{

// declaraciones locales

// bloque de código

[return] [expresión];

}

Unidad 2: Funciones definidas por el usuario

- declaraciones locales: declaración de constantes y/o variables. Tienen validez (ámbito) sólo dentro de la función.
- bloque de código: conjunto de instrucciones o sentencias que realizan la tarea específica de la función.
- [return] [expresión]: el valor que devuelve la expresión es el *"valor que retorna la función"*. Debe ser del mismo tipo indicado en la cabecera.

Si la función es **void** (no devuelve valor) puede omitirse la sentencia **return**. Una función puede tener más de un **return**.

- La función termina su ejecución cuando encuentra una sentencia **return** o la llave de cierre de la función.

Unidad 2: Funciones definidas por el usuario

3. Llamada:

- Se llaman con el nombre y la lista de argumentos.
- La función puede llamarse o invocarse desde cualquier otra función, excepto desde sí misma. Para este caso existen las funciones recursivas.

Unidad 2: Funciones – Biblioteca del usuario

- El usuario puede crear sus propias bibliotecas con las funciones por él definidas y que utiliza habitualmente.
- Uno de los procedimientos es el siguiente:
 1. Declarar, definir y llamar la función que desea guardar.
 2. Cuando la ejecución de la función sea correcta, copiar la definición de la función en un archivo y guardarlo en la carpeta de trabajo con extensión **.h** (por ej. **prueba.h**).
 3. Incluir en la compilación: **#include "prueba.h"**, de esta manera se pueden utilizar las funciones definidas en la biblioteca **prueba.h**

Unidad 2: Funciones – Biblioteca del usuario

Recordar que:

- **<prueba.h>** : el compilador busca el archivo en el directorio, de las bibliotecas, por defecto
- **"prueba.h"** : el compilador busca el archivo en el directorio de trabajo
- **"path"** : el compilador busca el archivo en la carpeta especificada.
 - **path:**
 - S.O. Windows : **"unidad:\\carpeta\\...\\prueba.h"**
 - S.O. Linux : **"unidad:/carpeta/.../prueba.h"**

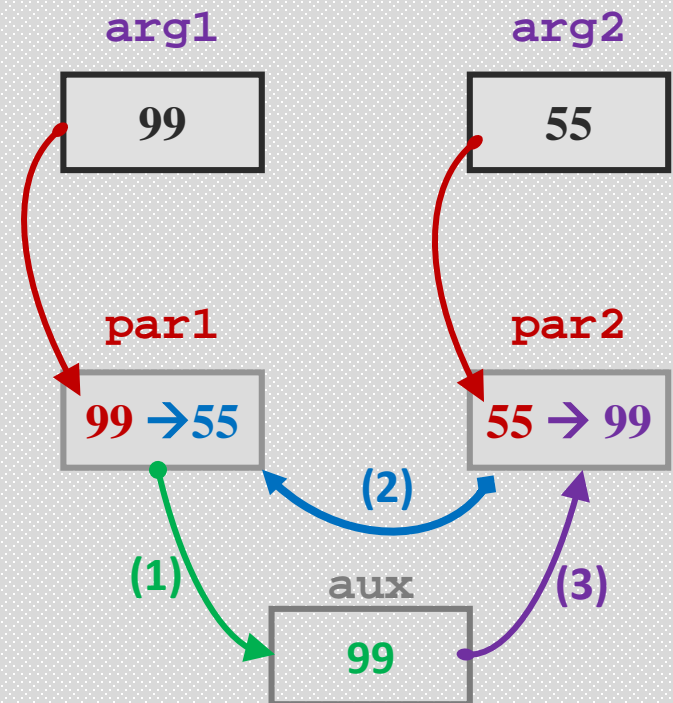
Unidad 2: Parámetros

Respecto al comportamiento de los parámetros de una función estos pueden recibir los valores desde los argumentos de dos formas: 1) por valor o 2) por referencia.

1. Parámetros por valor: los valores recibidos pueden ser modificados dentro de la función sin que afecten los valores iniciales de los respectivos argumentos. Se copian los valores desde los argumentos a los parámetros. No ocurre a la inversa.
2. Parámetros por referencia: las modificaciones que se realizan en los valores de estos parámetros SÍ modifican los valores de los argumentos de la llamada. Hay comunicación bidireccional entre los argumentos y los parámetros.

Unidad 2: Parámetros por valor

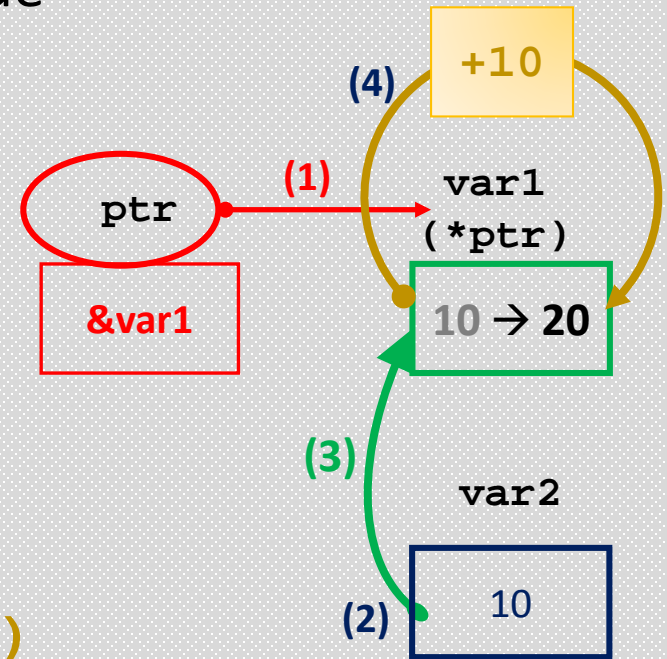
```
...  
void interc1(int par1, int par2)  
int main (int argc, char *argv[])  
{  
...  
int arg1 = 99, arg2 = 55;  
  
interc1(arg1, arg2);  
...  
}  
  
void interc1(int par1, int par2)  
{  
    int aux;  
    aux = par1;           // (1)  
    par1 = par2;          // (2)  
    par2 = aux;           // (3)  
}
```



Unidad 2: Funciones - Reseña de punteros

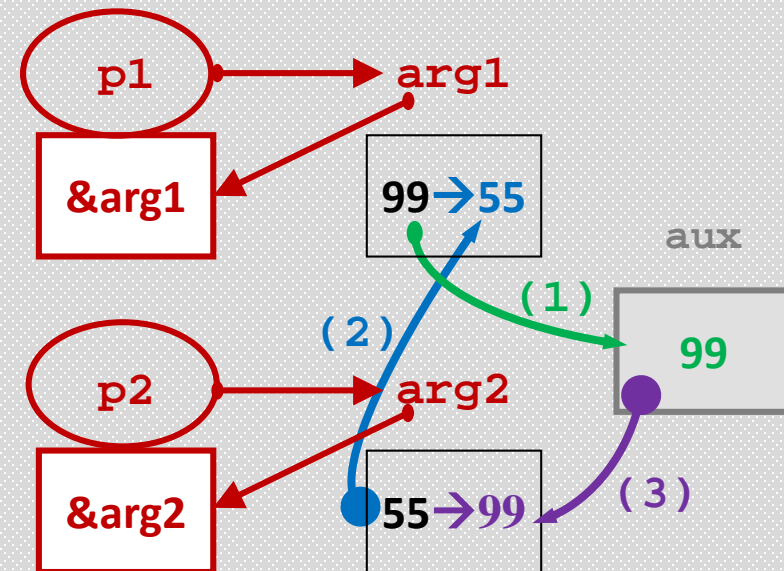
Antes de ver el tema de parámetros se verá un ejemplo con el uso de punteros a fin de afianzar los conceptos.

```
int var1, var2, *ptr;  
ptr = &var1; // (1)  
var2 = 10;   // (2)  
*ptr = var2; // (3)  
var1 = var1 + *ptr; //(4)
```



Unidad 2: Parámetros por referencia

```
...  
void interc2(int *p1, int *p2);  
int main (int argc, char *argv[])  
{  
...  
int arg1 = 99, arg2 = 55;  
  
interc2(&arg1, &arg2);  
...  
}  
  
void interc2(int *p1, int *p2)  
{  
    int aux;  
    aux = *p1;    // (1)  
    *p1 = *p2;    // (2)  
    *p2 = aux;    // (3)  
}
```



Unidad 2: Funciones en Lenguaje C - Ejemplos

1. Ingresar por teclado un número entero, devolver 0 si es par o uno si es impar.
2. Ingresar un número entero y devolver el caracter 'P' si es positivo, el caracter 'N' si es negativo y 'Z' si es cero.
3. Ingresar un número entero y con una sola función determinar si es múltiplo de 3 y de 4 simultáneamente.
4. Ingresar un número expresado en base 2. Usar una función para controlar que todos los dígitos sean binarios.
5. Ingresar un número decimal y convertirlo a base 2.

Unidad-2-Ej-09.c

Unidad-2-Ej-10.c

Unidad-2-Ej-11.c

Unidad-2-Ej-12.c

Unidad-2-Ej-13.c

Unidad 2: Funciones en Lenguaje C - Ejemplos

6. Calcular diferencia de potencial, intensidad de corriente o resistencia en un circuito. Utilizar funciones con menú.

Unidad-2-Ej-14.c

7. Número compuesto.

Unidad-2-Ej-15.c

8. Ingresar dos números expresados en base 2 y efectuar la suma en binario.

Unidad-2-Ej-16.c

9. Ingresar dos números expresados en base ***b*** (menor o igual que 10), controlar que se ingresaron correctamente los números y efectuar la suma en base ***b***.

Unidad-2-Ej-17.c

Unidad 2: Funciones en Lenguaje C

Almacenamiento y ámbito de las variables

Existen modificadores que afectan el modo que se almacenan las variables en el computador, estos son:

- 1.- **auto,**
- 2.- **static,**
- 3.- **volatile,**
- 4.- **register,**
- 5.- **extern.**

Unidad 2: Funciones en Lenguaje C

Almacenamiento y ámbito de las variables

1.- **auto** (automáticas):

a) Por defecto todas las variables locales son automáticas.

b) Características

i. Ámbito local: sólo tienen validez en la función donde fueron declaradas.

ii. Almacenamiento local:

- no necesariamente se inician en cero en tiempo de compilación,
- pierden el valor asignado cuando se termina la ejecución de la función donde fueron declaradas y utilizadas.

Unidad 2: Funciones en Lenguaje C

Almacenamiento y ámbito de las variables

2.- static (estáticas). Las características son:

- i. Ámbito local: sólo tienen validez en la función donde fueron declaradas.
- ii. Almacenamiento global:
 - i. se almacenan en la zona del segmento de datos de la memoria,
 - ii. se inicializan en cero en tiempo de compilación,
 - iii. mantienen sus valores hasta la finalización del programa.

Unidad 2: Funciones en Lenguaje C

Almacenamiento y ámbito de las variables

3. **volatile**(volátil): Indica al compilador que el valor de la variable puede cambiar por otra instrucción que no sea la de asignación.

4. **register**:

- a) Se almacenan en un registro del microprocesador.
- b) No tienen dirección de memoria.
- c) La instrucción **&x** es inválida si **x** es una variable con el modificador **register**.
- d) Si los registros están ocupados, la variable se guarda en la memoria pero no hay notificación.

Unidad-2-Ej-21.c

5. **extern** (externa): Especifica que la variable está declarada en otro archivo perteneciente al proyecto.

Unidad 2: Funciones en Lenguaje C

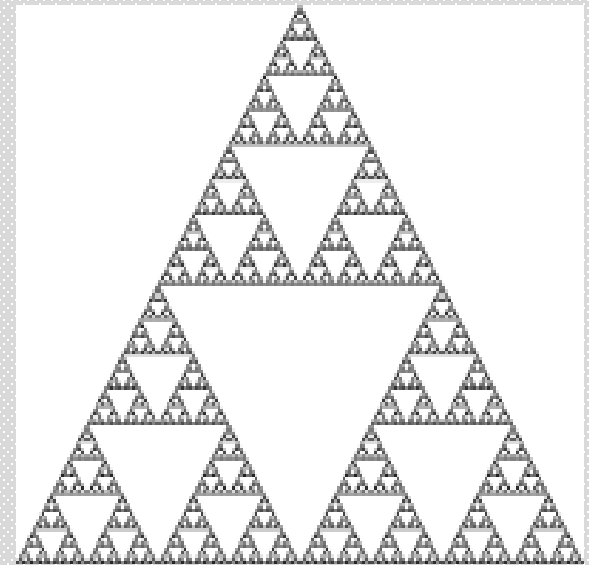
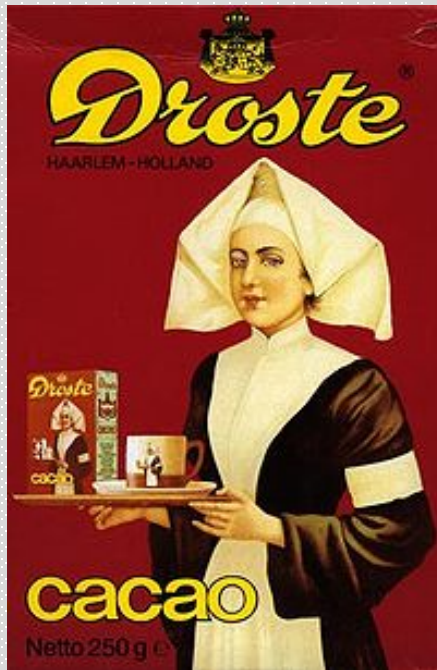
Recursividad

- Es la forma en la cual se especifica un concepto o proceso basado en su propia definición.
- Para que una definición recursiva esté completamente identificada es necesario tener un caso base que no se calcule utilizando casos anteriores y que la división del problema converja a ese caso base.

Unidad 2: Funciones en Lenguaje C

Recursividad

Imágenes recursivas



Unidad 2: Funciones en Lenguaje C

Funciones recursivas

- Es aquella función que puede llamarse o invocarse a sí misma.
- Debe tener un "caso base" o "solución trivial".
- Los casos bases son aquellos que se resuelven con un segmento de código sin recursividad.
- Los casos generales siempre deben avanzar hacia el caso base del problema.

Unidad 2: Funciones en Lenguaje C

Funciones recursivas

Producto: $x * n$

$x * n \rightarrow$ sumar n veces x

$$x * n = x + x + x + x + x + \dots + x$$

$$x * n = x + x * (n-1)$$

$$x * n = x + x + x * (n-2)$$

$$x * n = x + x + x + x * (n-3)$$

\dots

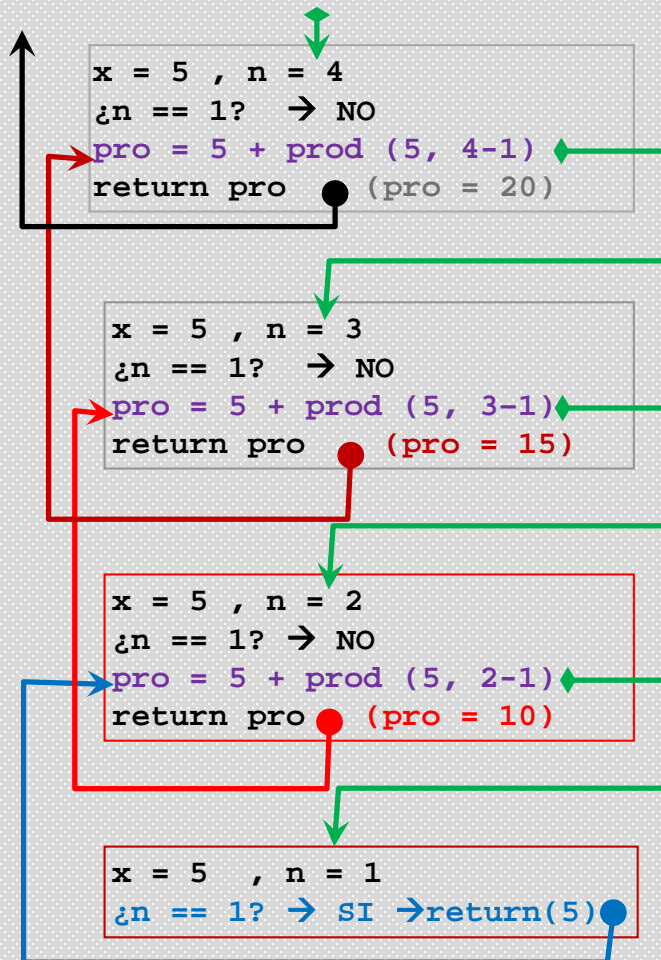
$$x * n = x + x + x + x + \dots + x * (n - (n-1))$$

$$x * n = x + x + x + x + \dots + x * 1$$

Unidad 2: Funciones en Lenguaje C

Funciones recursivas

return(20) prod(5, 4)



...

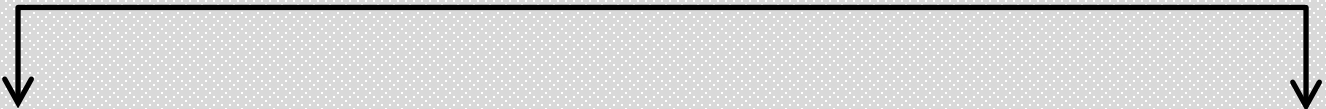
```
int prod (int x, int n)
{
    int pro;
    if (n == 1) //caso base
        return x;
    else
        pro = x + prod(x, n-1);
    return pro;
}
```


Unidad 2: Funciones en Lenguaje C

Funciones recursivas

Factorial: $n!$

producto de n a 1


$$n! = n * (n-1) * (n-2) * (n-3) * \dots * (n - (n-1))$$

$$n! = n * (n-1) * (n-2) * (n-3) * \dots * 1$$


$$n! = n * (n-1)!$$

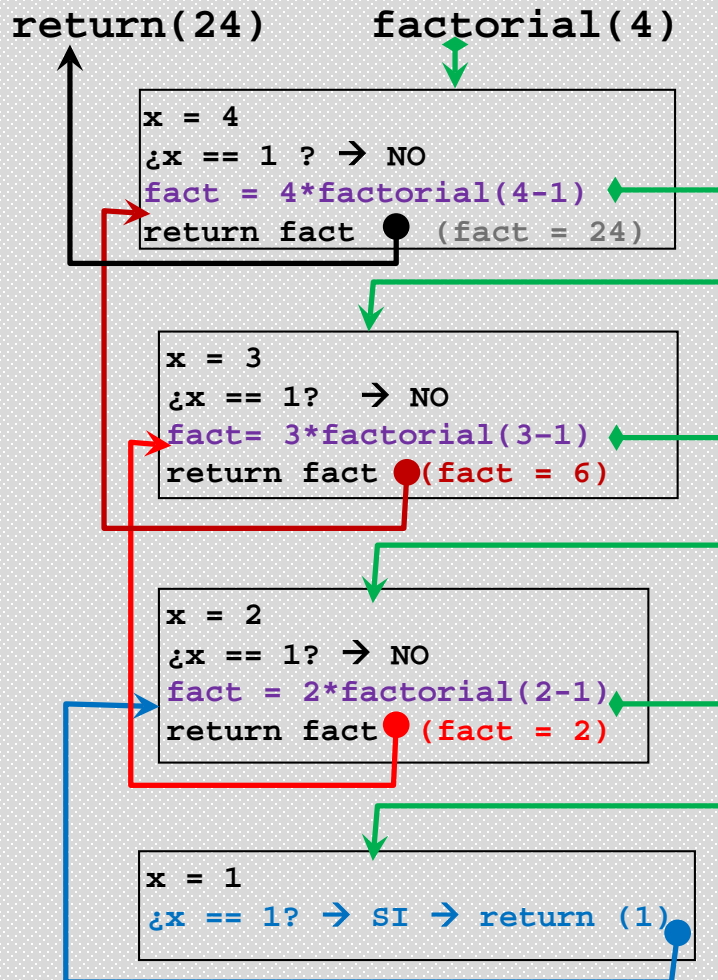
$$n! = n * (n-1) * (n-2)!$$

$$n! = n * (n-1) * (n-2) * (n-3)!$$

$$n! = n * (n-1) * (n-2) * \dots * 1!$$

Unidad 2: Funciones en Lenguaje C

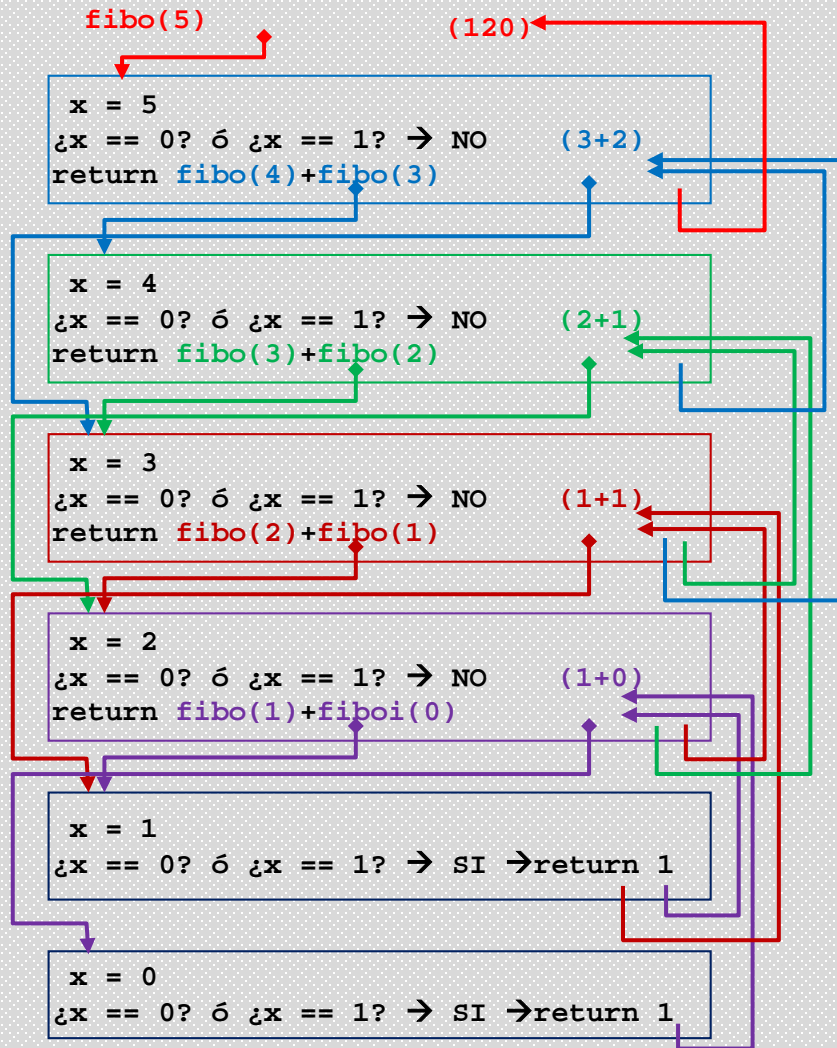
Funciones recursivas



```
...  
int factorial (int x)  
{  
    int fact;  
    if (x == 1)  
        return 1; // caso base  
    else  
        fact = x*factorial(x-1);  
    return fact;  
}
```

Unidad 2: Funciones en Lenguaje C

Funciones recursivas



...

```
printf("%d", fibo(5));
```

...

```
int fibo( int n )
```

```
{
```

```
    //caso base
```

```
    if ( n == 0 || n == 1 )
```

```
        return n;
```

```
    else
```

```
        return fibo(n-1)+fibo(n-2);
```

```
}
```

Unidad 2: Funciones en Lenguaje C

Argumentos de la función main()

```
...  
int main(int argc, char *argv[])  
{  
    ...  
    // bloque de código  
    ...  
    return 0;  
}
```

argc: toma número de argumentos recibidos por el programa. El nombre del programa también se cuenta como argumento.

***argv**: es un puntero a un arreglo de `char` que contiene los parámetros pasados en el mismo orden en que fueron escritos.