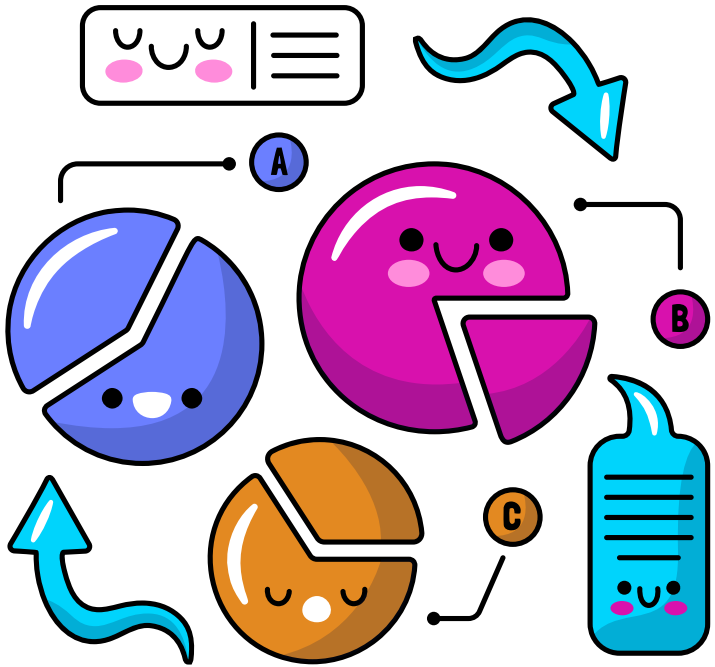
Funciones

pasaje de parámetros

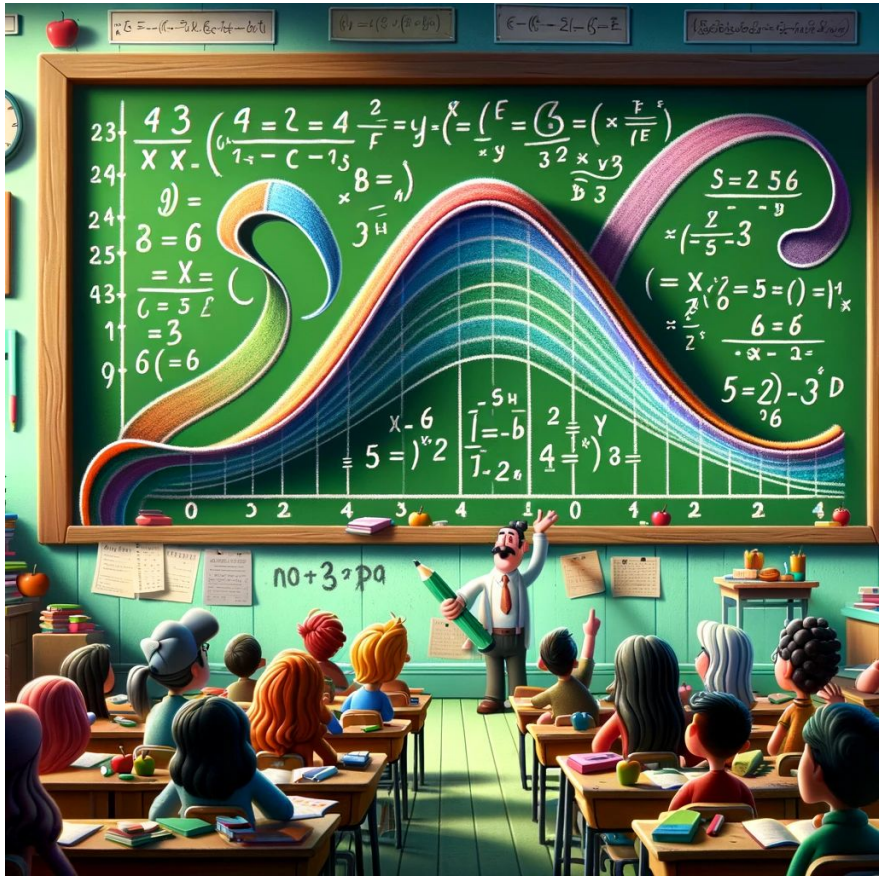


Repaso

-Estructuras de Control:
    if else
    while
    for
    Bloque {}

-Entrada y Salida de Datos

Funciones



Funciones: Definición

``una función es simplemente una máquina, que yo le voy a dar una orden.''

que es una función

Funciones: Definición

``Las funciones **dividen tareas grandes de computación** en varias **más pequeñas** y, permiten la posibilidad de **construir sobre lo que otros ya han hecho**, en lugar de comenzar desde cero.... C ha sido diseñado para hacer que las funciones sean eficientes y fáciles de usar. Generalmente los programas en C consisten en muchas funciones pequeñas en lugar de unas pocas grandes.''

Funciones: Definición

Una función conceptualmente es la **unidad de abstracción** de un programa.

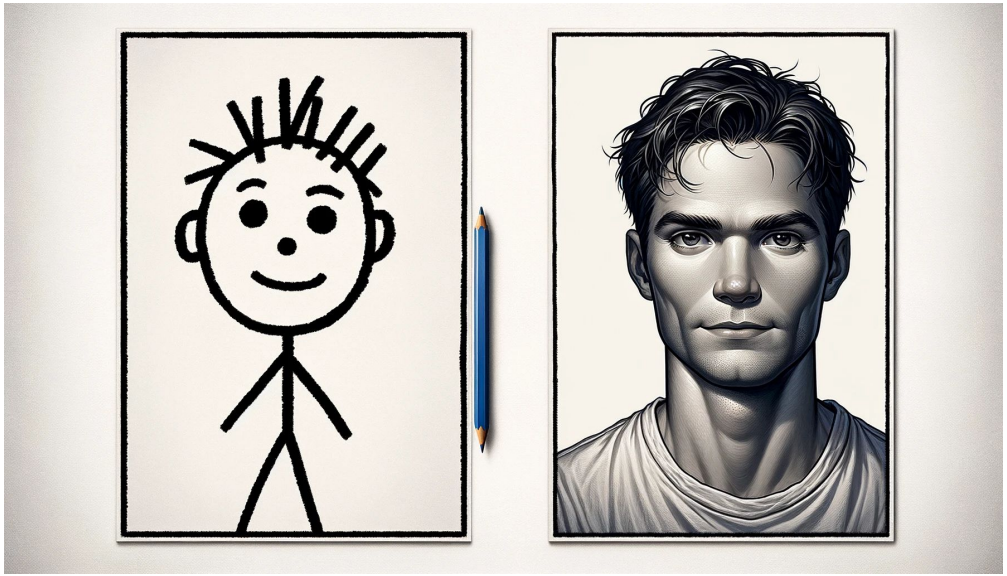
Una función debe describir **una acción atómica** dentro de un programa.

Conceptualmente son la herramienta que permite utilizar el principio de **Divide y Conquista** (divide et impera!).

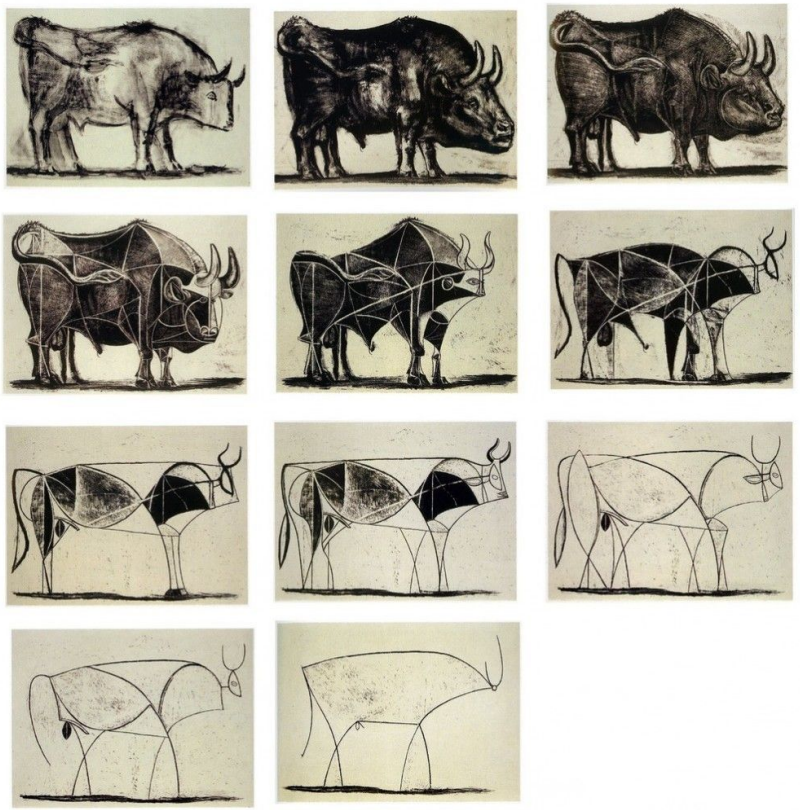
Dividen el programa en **piezas manejables** y r**educen la duplicación** de código fuente.

Funciones: Abstracción

La abstracción es el proceso por el cual un problema se despoja de todos aquellos aspectos no relevantes para que este pueda ser solucionado



Funciones: Abstracción

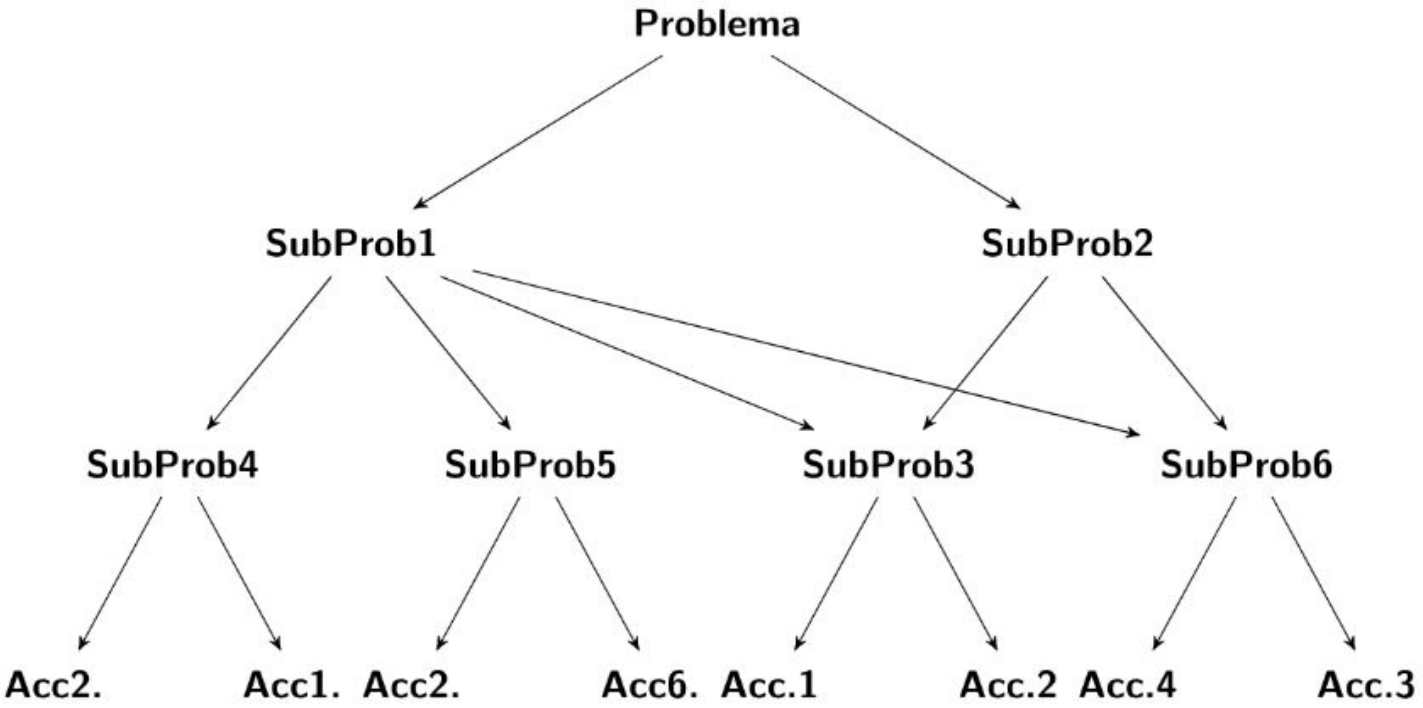


Funciones: Modularización

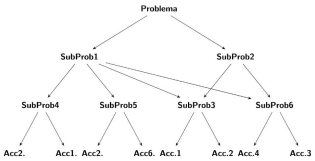
Unas de las técnicas de resolución de problemas se basa la identificación de sub-problemas recurrentes, de forma tal que estos sean más fáciles de resolver que el problema en todo su conjunto.

Dicha identificación de sub-problemas se realiza de forma tal que la solución de los sub-problemas encontrados se trivial, también llamadas acciones.

Funciones: Programación Top-Down

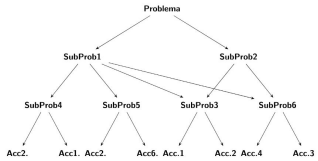


Funciones: Programación Top-Down



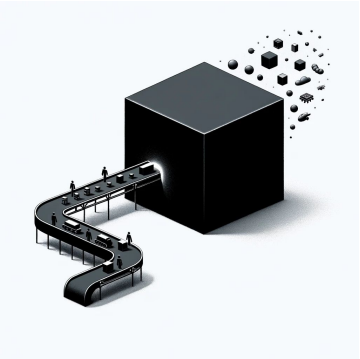
Se basa la construcción de un programa a partir de una especificación general o de alto nivel de lo que el mismo debe hacer, posteriormente se descompone esta especificación en piezas más y más sencillas hasta alcanzar un nivel que corresponda a las **acciones primitivas** del lenguaje en el cual se implementar á el programa.

Funciones: Programación Top-Down



Una ***acción primitiva***, es toda aquella acción que puede ejecutarse sin necesidad de alg ún tipo de explicación para aquel que la ejecuta. Se debe tener en cuenta que una acción primitiva en un lenguaje de programación, puede no serlo en otro.

-La ***descomposición o la partición***.
-Los refinamientos sucesivos.

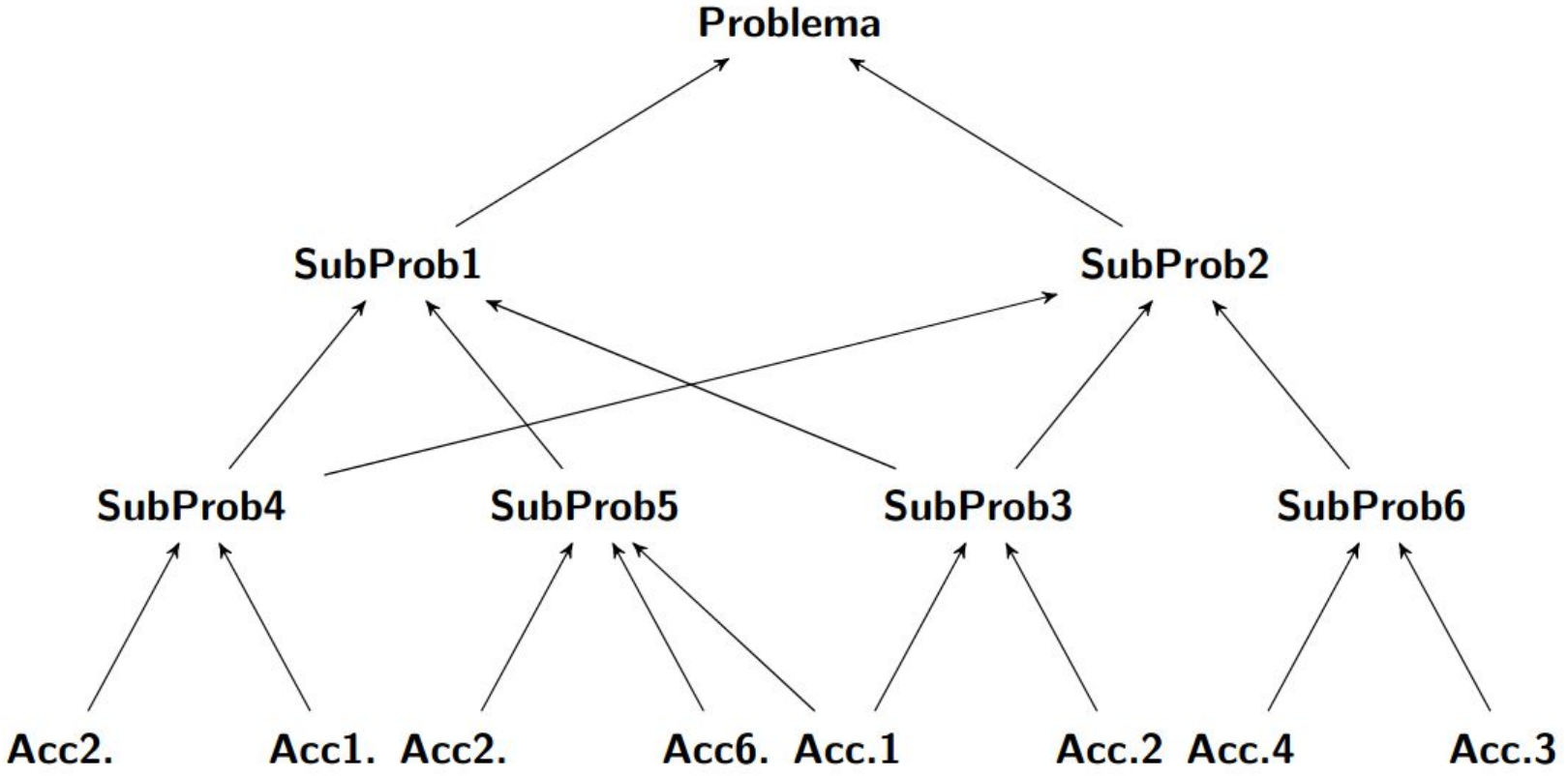


Funciones: Programación Bottom-Up

La programación **Bottom-up o ascendente** por el contrario es una técnica de programación que a partir de acciones primitivas del lenguaje **construye poco a poco acciones cada vez más complejas** hasta llegar a obtener todas las necesarias para la construcci ón del program.

La idea es tener a priori las acciones y a partir de esa colección de acciones crear la solución hacia arriba.

Funciones: Programación Bottom-Up



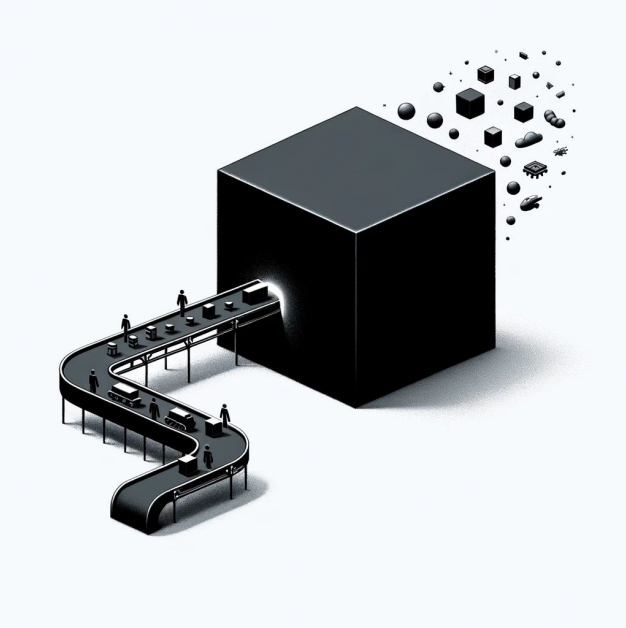
Funciones: Programación Bottom-Up

Esta técnica de programación sustenta sus principios en :

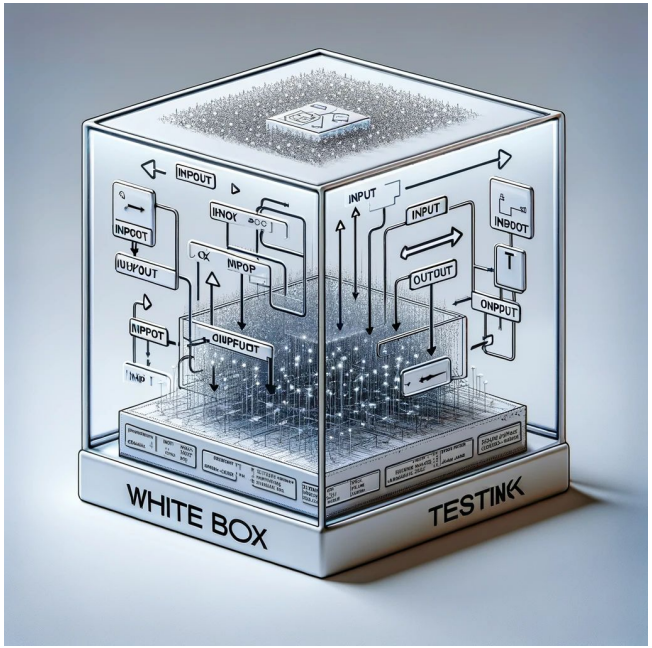
1. La composición.

2. En el concepto de “hágalo usted mismo!” (por ejemplo un mueble comprado en un supermercado).

Funciones



Top-Down



Bottom-Up

Funciones: en C

En cualquier lenguaje de programación las funciones se crean mediante un mecanismo llamado **declaración de funciones** :

```
tipo_retorno nombre_funcion ( lista de parametros ) {
        declaraciones acciones
}
```

Funciones: en C

```
tipo_retorno nombre_funcion ( tipo_1 Par_1 , tipo_2 Par_2 , ... , tipo_N Par_N ) {

/* declaraciones */

/* acciones */
     accion_1 ;
     accion_2 ;
     accion_3 ;
     accion_4 ;
     accion_5 ;
     . . . .
     accion_N ;

}
```

Funciones: en C

Tipo de retorno: El tipo de retorno corresponde al tipo de dato del valor que devolverá la función tras ser ejecutada.
La acción que permite la ejecución de una función se denomina invocaci ón. Existen dos roles entre una función y aquel que la utiliza. El rol de la parte del programa que hace uso de una función se llama "***invocador o llamador***".

El rol de la función al ser utilizada por alguna parte del programa se denomina "***invocada o llamada***". Si a una función
no se le asigna tipo de retorno esta devolveŕa por defecto un valor de tipo entero. La instrucción que se encarga de
devolver el valor de la función y devolver el control al invocador es **return**.

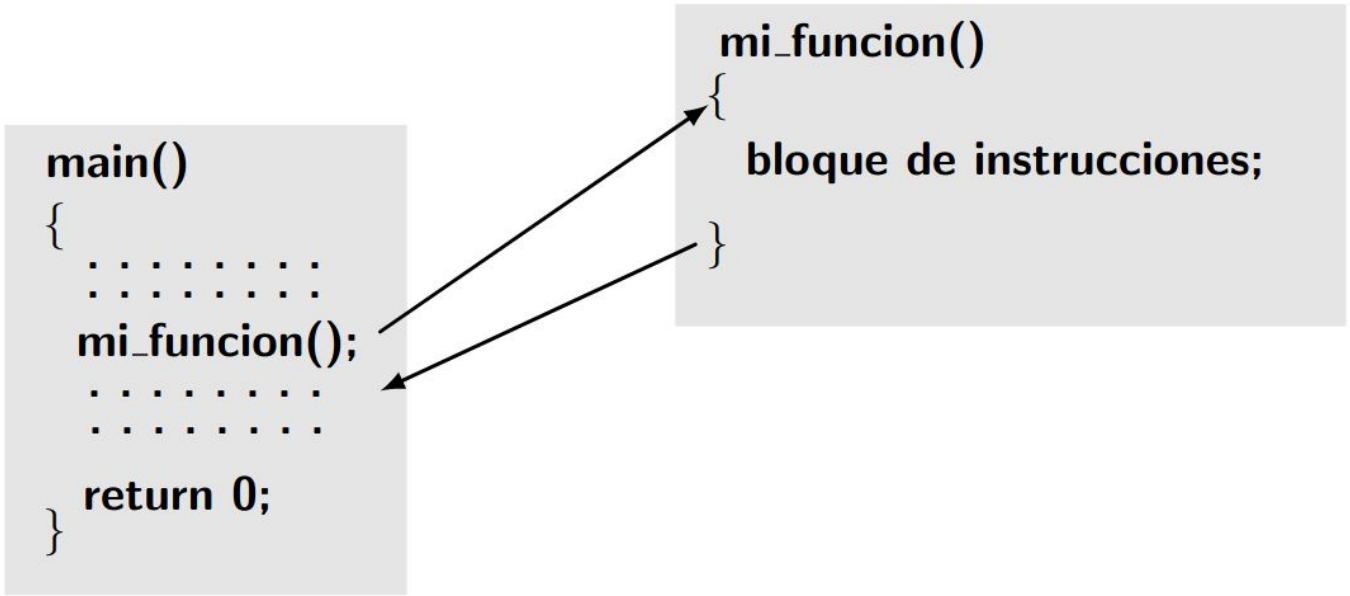
Funciones: en C

Nombre de la funci ón: Toda funci ón debe tener un nombre que cumpla con las reglas sint ácticas de los identificadores válidos en C.

Lista de par ámetros: Un parámetro es una variable utilizada para recibir valores de entrada en una funci ón. La lista de par ámetros consiste en una lista separada por comas que contiene las declaraciones de los par ámetros recibidos por la función al ser **invocada**.

Declaraciones y Acciones: Dentro del cuerpo de una función se espera encontrar declaraciones de variables, necesarias para alcanzar el objetivo de la función. Estas variables declaradas dentro de la funci ón se denominan *variables locales*, una variable es local dentro del bloque de programa, en este caso una función, en la cual es declarada. También se espera encontrar acciones y estructuras de control.

Funciones: en C



Funciones: en C

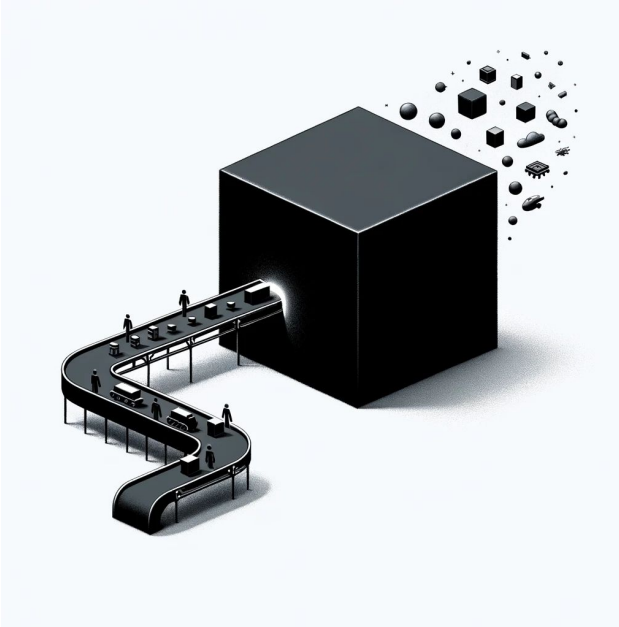
```c
# include <stdio.h>

int cuadrado (int numero){
    return numero*numero;
}

int main(){
     int i;

     printf("\n");
     for ( i=1 ; i<=10 ; i++){
          printf ( "%d  ", cuadrado(i));
    }
    printf("\n\n");

    return 0;
}
```

Funciones: Como Caja Negra

La visión de caja negra hace que veamos a una función como a algo que recibe entradas y produce salidas.

Las entradas son los parámetros

Las Salidas son el valor de retorno



Funciones: Parámetros

Un parámetro es una variable utilizada para recibir valores de entrada en una función. La lista de par ámetros consiste en una lista separada por comas que contiene las declaraciones de los parámetros recibidos por la función al ser invocada.

un función matemática puede depender de distintas variables por ejemplo:

$$f(x,y) = 3x^2 + 2y - 3xy$$

x e ***y*** son dos variables que funcionan como parámetros dados un par (***x,y***) se obtiene un valor para **f(x)**.

Al igual que en matemática las funciones en los lenguajes de programación reciben valores (parámetros) y devuelven su correspondiente valor al ser aplicados a la función.

Funciones: Parámetros

Dado que C es un lenguaje tipado, es necesario informar a la función que tipo de parámetros recibe. Normalmente las funciones reciben una lista de parámetros.

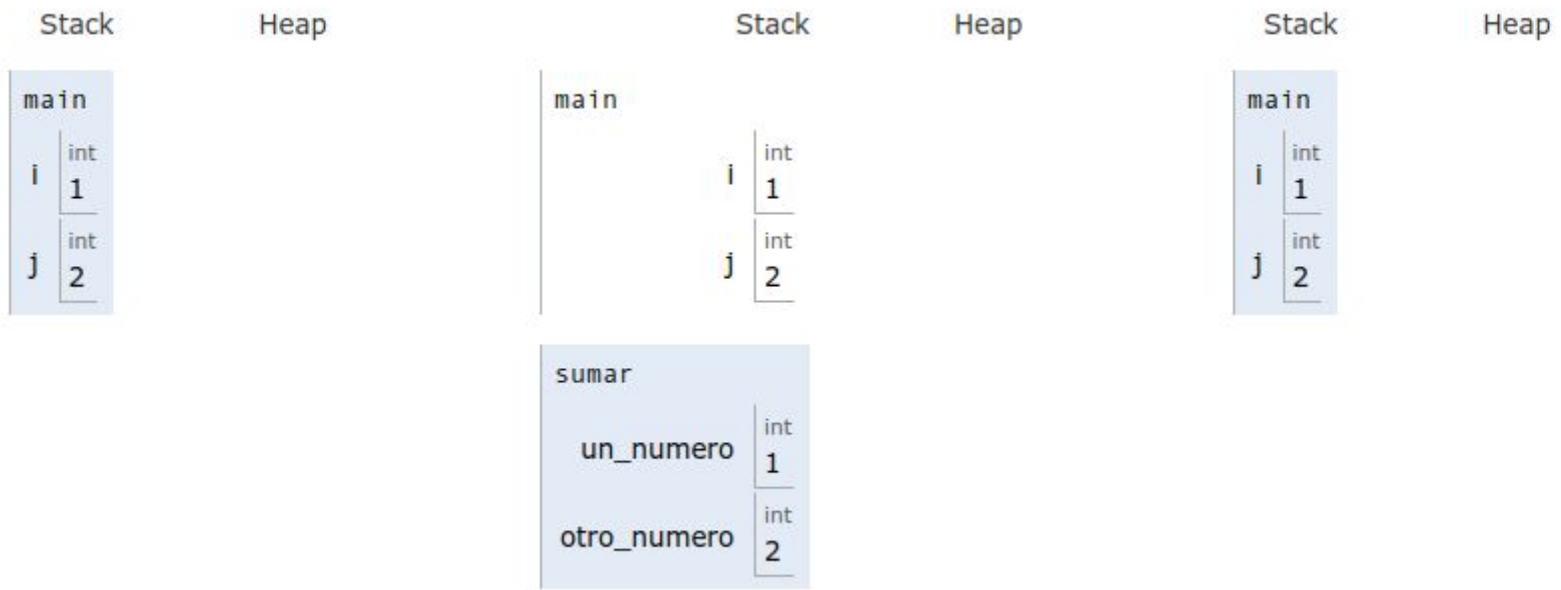
Ejemplo: Función que sume dos números enteros.

```
int sumar (int un_numero, int otro_numero){
    return un_numero + otro_numero;
}
```

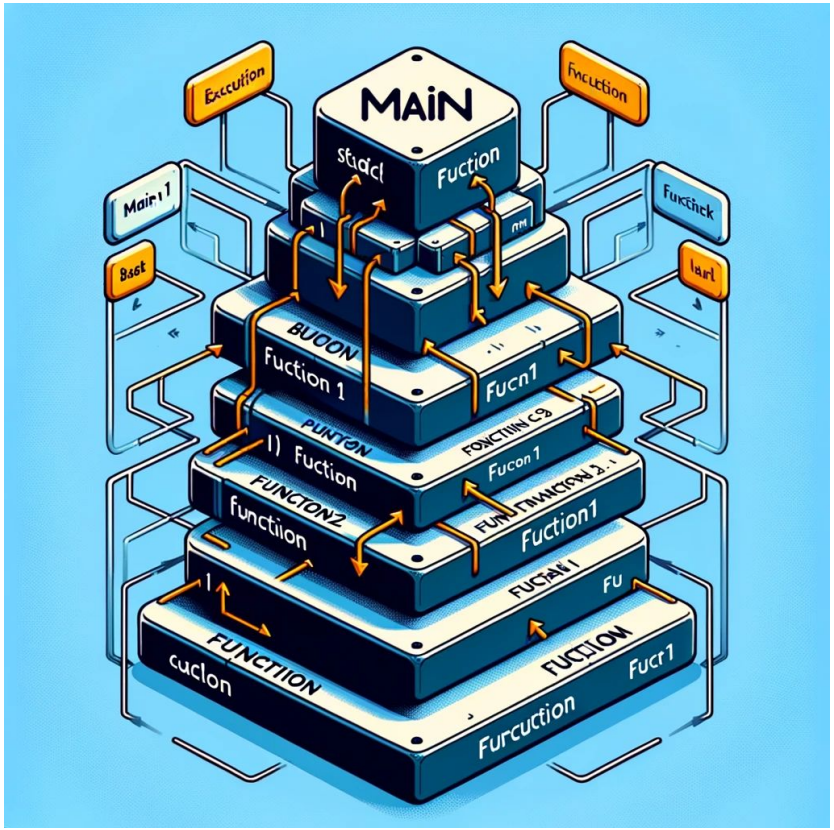
```
la llamada a esta función sería sumar(1,1)
```

Funciones: Parámetros por Valor

Qué está pasando?



Funciones: el Stack de Ejecución



Funciones: el Stack de Ejecución

- La **pila de ejecución** mantiene un registro del estado de las funciones activas en un programa.
- Cada **llamada a función crea un nuevo marco de pila** (a veces llamado marco de activación o registro de activación) que contiene los valores de sus parámetros y variables locales.
- El marco **en la parte superior de la pila es el marco activo**; representa la activación de la función que actualmente se está ejecutando, y solo sus variables locales y parámetros están en contexto.
- Cuando se llama a una función, **se crea un nuevo marco de pila para ella (se coloca en la parte superior de la pila) y se asigna espacio para sus variables locales y parámetros en el nuevo marco.** Cuando una función retorna, **su marco de pila se elimina de la pila** (se saca de la parte superior de la pila), dejando el marco de pila del llamador en la parte superior de la pila.

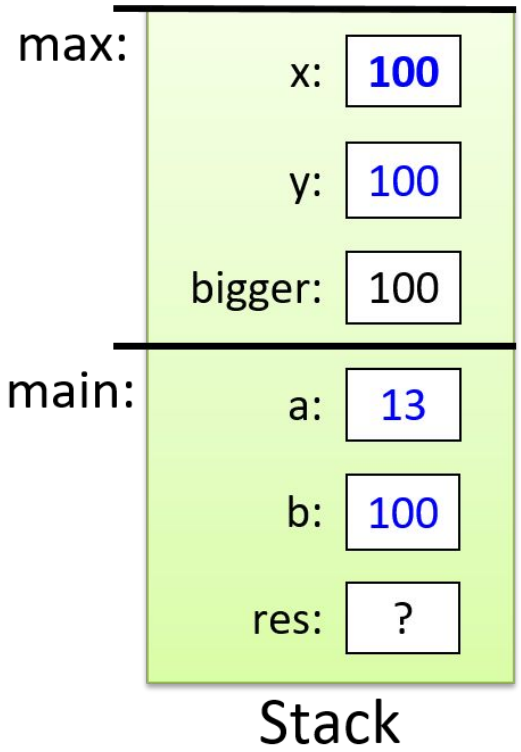
Funciones: el Stack de Ejecución

```
#include <stdio.h>
int main(void) {
    int a=13, b=10, res;

    res = max(a, b);
    printf("el maximo es is %d\n" res);
    return 0;
}
```

```
int max(int x, int y)  {
    int result;

    result = x;
    if (y > x) {
        result = y;
    }
    return result;
}
```

Funciones: el Stack de Ejecución

```
#include <stdio.h>
int main(void) {
    int a=13, b=10, res;
    ...
    res = max(a, b);
    ...
  }


 int max(int x, int y)
```



Funciones: Sin Valor de Retorno

En muchos lenguajes de programación existe el concepto de procedimiento, este es comparable a una función que no retorna valor alguno. En C, no existen los procedimientos como una estructura separada de las funciones.
Para definir lo que en otros lenguajes de programación es conocido como un procedimiento en C se utiliza una función con valor de retorno nulo.

Para ello existe el tipo de dato **void**. Este tipo de dato utilizado como valor de retorno de una función indica que la misma no devuelve valor alguno.

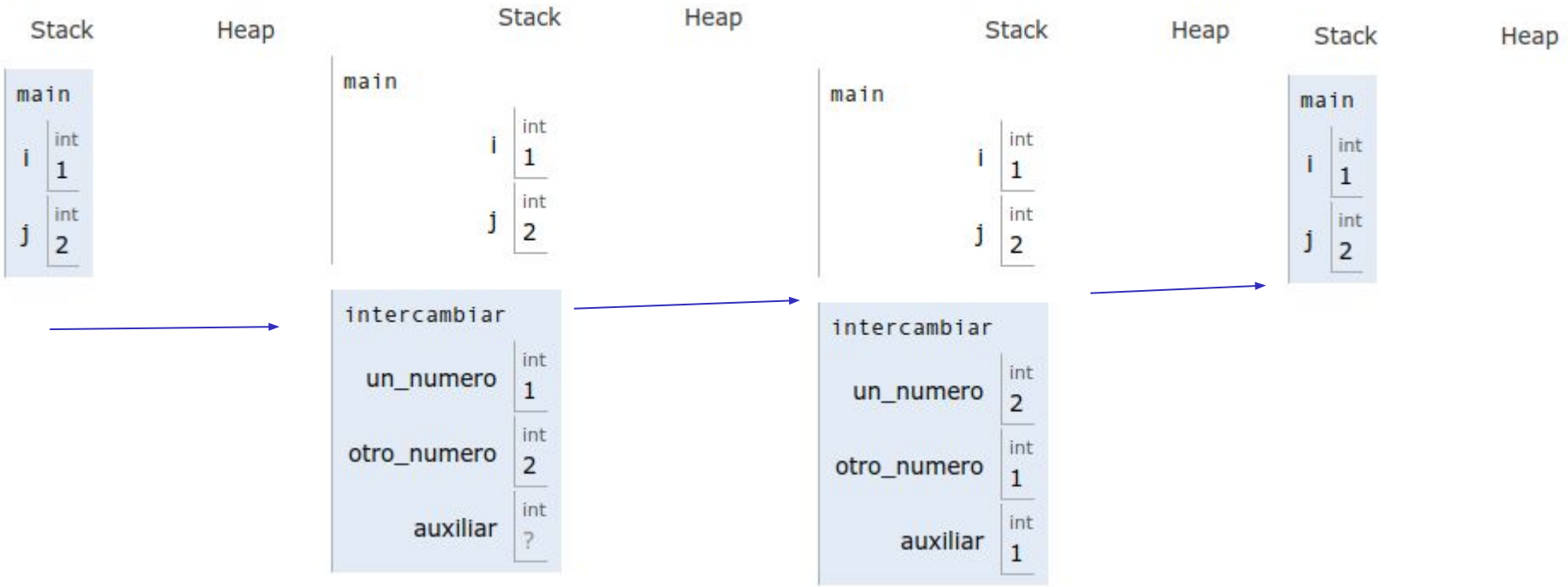
También puede ser usado en la lista de parámetros formales, para indicar que la función no recibe parámetros:

void un_procedimiento (parametro_1 , parametro_2 ,... , parametro_N)

Funciones: Pasaje de Parámetros por Valor

```
# include <stdio.h>

void intercambiar (int un_numero, int otro_numero){
        int auxiliar=0;

        auxiliar=un_numero;
        un_numero=otro_numero;
        otro_numero=auxiliar;
}

int main(){
        int i=1,j=2;

        printf ( "i:%d  j:%d\n", i,j);
        intercambiar(i,j);
        printf ( "i:%d  j:%d\n", i,j);
        return 0;
}
```

Funciones: Pasaje de Parámetros por Referencia



Funciones: Pasaje de Parametros por Referencia

```
# include <stdio.h>

void intercambiar (int* un_numero, int* otro_numero){
    int auxiliar=0;

    auxiliar=*un_numero;
    *un_numero=*otro_numero;
    *otro_numero=auxiliar;
}
int main(){
    int i=1,j=2;

    printf ( "i:%d  j:%d\n", i,j);
    intercambiar(&i,&j);
    printf ( "i:%d  j:%d\n", i,j);
    return 0;
}
```