



Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

**Unidad Profesional Interdisciplinaria de
Ingeniería Campus Tlaxcala UPIIT**

Fundamentos de Programación

Esaú Eliezer Escobar Juárez

Estructura de los programas que hemos visto hasta el momento

```
#include<archivo_cabecera>

int main() {
    declaración_variables_locales;
    instrucción_1;
    instrucción_1;
    . . .
    instrucción_1;
    return 0;
}
```

- Se tiene un solo archivo fuente
- Todo lo que hace el programa se implementa dentro de la función main.

PROBLEMAS DE ESTA METODOLOGIA

```
#include <stdio.h>
```

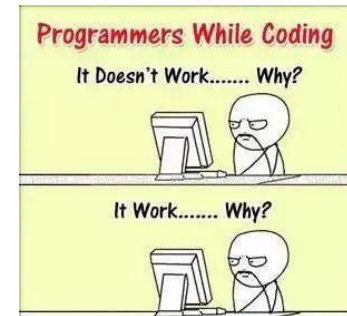
```
int main() {  
    int a[3],b[3],c[3];  
    int i;  
    printf(" Ingrese el primer arreglo: ");  
    for(i=0;i<3;i++) {  
        scanf("%d",&a[i]);  
    }  
    printf(" arr1 = [%d %d %d]\n\n",a[0],a[1],a[2]);  
    printf(" Ingrese el segundo arreglo: ");  
    for(i=0;i<3;i++) {  
        scanf("%d",&b[i]);  
    }  
    printf(" arr2 = [%d %d %d]\n\n",b[0],b[1],b[2]);  
    for(i=0;i<3;i++) {  
        c[i]=a[i]+b[i];  
    }  
    printf(" arr3 = arr1 + arr2 = [%d %d %d]\n\n",c[0],c[1],c[2]);  
    return 0;  
}
```

Repetición de código

Código repetitivo

PROBLEMAS DE ESTA METODOLOGIA

- Programación repetida.
- A medida que el problema se hace mas complejo el main tiende a crecer mucho y se hace más difícil de entender.
- Dificultad para programar.
- Poca reutilización de código.



SOLUCION AL PROBLEMA. FUNCIONES



**División del
trabajo**



EJEMPLO

Realizar un programa que calcule el valor de la función seno usando la serie equivalente para ello. Los valores de entrada son x y el numero de términos. A continuación se muestra la serie equivalente:

$$\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

SOLUCION SIN HACER USO DE FUNCIONES

```
...
den = 1;
for(i=1;i<=N;i++){
    fac = 1;
    pot = 1;
    for(j=1;j<=den;j++){
        fac = fac*j;
        pot = pot*x;
    }
    term = pot/fac;
    if(signo==0){
        resul += term;
        signo = 1;
    }
    else{
        resul -= term;
        signo = 0;
    }
    den+=2;
}
...
```

Ciclo para generar el numero de términos que tendrá la serie

Ciclo anidado para calcular el factorial (que se usara en el denominador) y la potencia (que se usara en el numerador).

Parte que sumara o restara el termino (term) al acumulador (resul) usado para almacenar la solución.

$$\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

resul

term

SOLUCION AL PROBLEMA: FUNCIONES

```
#include <archivo_cabecera>

int main() {
    declaracion_variables_locales;
    instruccion_1;
    instruccion_2;
    . . .
    instruccion_N;
    return 0;
}
```



```
#include <archivo_cabecera>

protipo_funciones;

int main() {
    declaracion_variables_locales;
    funcion_1();
    funcion_2();
    . . .
    funcion_N();
    return 0;
}

funcion_1() {
    codigo_funcion_1;
}

. . .

funcion_N() {
    codigo_funcion_N;
}
```



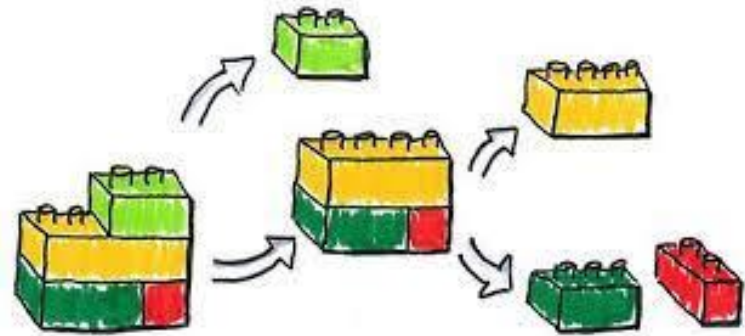
SOLUCION HACIENDO USO DE FUNCIONES

```
...  
den = 1;  
for(i=1; i<=N; i++){  
    fac = potencia(x, den);  
    pot = factorial(den);  
    term = pot/fac;  
    if(signo==0){  
        resul += term;  
        signo = 1;  
    }  
    else{  
        resul -= term;  
        signo = 0;  
    }  
    den+=2;  
}  
...
```

Ciclo para generar el numero de términos que tendrá la serie

Calculo del factorial y la potencia usando funciones.

Parte que sumara o restara el termino (term) al acumulador (resul) usado para almacenar la solución.



SOLUCION HACIENDO USO DE FUNCIONES

```
...
den = 1;
for(i=1;i<=N;i++){
    fac = potencia(x,den);
    pot = factorial(den);
    term = pot/fac;
    if(signo==0){
        resul += term;
        signo = 1;
    }
    else{
        resul -= term;
        signo = 0;
    }
    den+=2;
}
...
```

```
...
den = 1;
for(i=1;i<=N;i++){
    fac = 1;
    pot = 1;
    for(j=1;j<=den;j++){
        fac = fac*j;
        pot = pot*x;
    }
    term = pot/fac;
    if(signo==0){
        resul += term;
        signo = 1;
    }
    else{
        resul -= term;
        signo = 0;
    }
    den+=2;
}
...
```

SOLUCION HACIENDO USO DE FUNCIONES

```
...  
den = 1;  
for(i=1;i<=N;i++){  
    fac = potencia(x,den);  
    pot = factorial(den);  
    term = pot/fac;  
    if(signo==0){  
        resul += term;  
        signo = 1;  
    }  
    else{  
        resul -= term;  
        signo = 0;  
    }  
    den+=2;  
}  
...
```

```
float potencia(float x, int y){  
    int i;  
    float pot = 1;  
    for(i=1;i<=y;i++){  
        pot *= x;  
    }  
    return pot;  
}
```

```
long factorial(int x){  
    int i;  
    long fac = 1;  
    for(i=1;i<=x;i++){  
        fac *= i;  
    }  
    return fac;  
}
```

SOLUCION AL PROBLEMA:

FUNCIONES

```
#include <stdio.h>

int a[3],b[3],c[3];

void ingresar_arreglo(int []);
void imprimir_arreglo(int []);
void sumar_arreglos(int [],int [],int*);

int main() {
    int i;
    printf(" Ingrese el primer arreglo: ");
    ingresar_arreglo(a);
    printf("arr1 = ");
    imprimir_arreglo(a);
    printf("\n\n");
    . . .
    return 0;
}

void ingresar_arreglo(int arr[]) {
    for(i=0;i<3;i++) {
        scanf("%d",&arr[i]);
    }
}

void imprimir_arreglo(int arr[]) {
    printf("[%d %d %d]",arr[0],arr[1],arr[2]);
}

void sumar_arreglos(int a1[],int a2[],int *res){
    for(i=0;i<3;i++) {
        res[i]=a1[i]+a2[i];
    }
}
```

Prototipos de las funciones

Función principal

Definición de las funciones

INTRODUCCION A LAS FUNCIONES

- El uso de funciones permite dividir grandes tareas.
- Se ahorra programación repetida.
- Se evita reinventar la rueda gracias a la reutilización de código.
- Hace que los programas sean mas modulares, mas fáciles de leer y mas fáciles de editar .



CONCEPTOS PREVIOS

Ambito: Área del programa en la cual la variable es válida (existe).

Global: Válida en todo el programa.

Local: Su ámbito es limitado al bloque en el cual se declara y no puede ser accedida fuera de dicho **bloque**.

Sección de código encerrada entre llaves ({}).

```
#include <stdio.h>
#include <stdlib.h>
```

```
int a, b = 1, c = 2;
```

```
int main() {
```

```
    int d = 3, e = 3;
```

```
    {
```

```
        int f = 8;
```

```
        int g = 9;
```

```
    }
```

```
    return 0;
```

```
}
```

Variables globales: a, b, c

Variables locales: d, e, f, g

CONCEPTOS PREVIOS

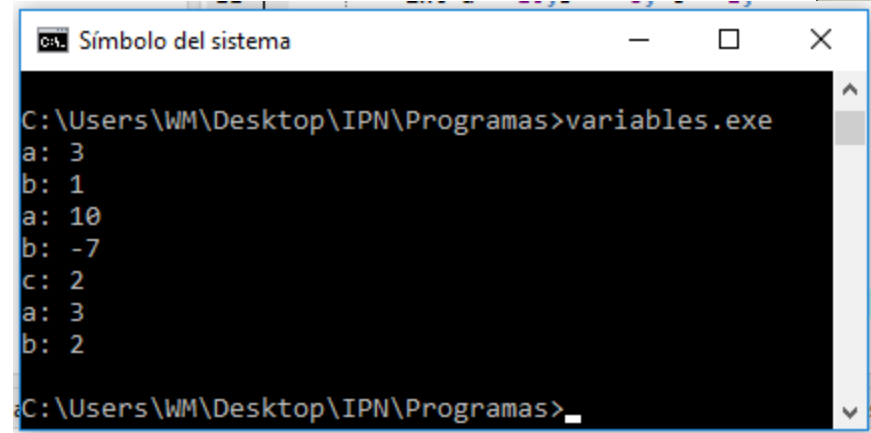
Ocultamiento de variables: Se da cuando se declara una variable con el mismo nombre de una variable global (o de otra local con un ámbito mayor). Básicamente lo que sucede es que el valor del ámbito mas interno prima sobre el valor del ámbito mas externo.

```
#include <stdio.h>
#include <stdlib.h>
```

```
int a, b = 1;
int main() {
    int a = 3;
    printf("a: %d\n", a);
    printf("b: %d\n", b);
    b++;
    {
        int a = 10, b = -8, c = 2;
        b++;
        printf("a: %d\n", a);
        printf("b: %d\n", b);
        printf("c: %d\n", c);
        a+=8;
    }
    printf("a: %d\n", a);
    printf("b: %d\n", b);
    return 0;
}
```

Ocultamiento
de la variable
global a.

Ocultamiento
de las
variables
locales a y b.



```
Símbolo del sistema
C:\Users\WM\Desktop\IPN\Programas>variables.exe
a: 3
b: 1
a: 10
b: -7
c: 2
a: 3
b: 2
C:\Users\WM\Desktop\IPN\Programas>
```

CONCEPTOS PREVIOS

Almacenamiento de variables

Permanente: El valor de la variable se mantiene (no se borra) una vez que esta ha sido inicializada. Cuando el almacenamiento es permanente la inicialización solo se realiza una vez. Para especificar almacenamiento permanente se usa la palabra **static** en la declaración de la variable.

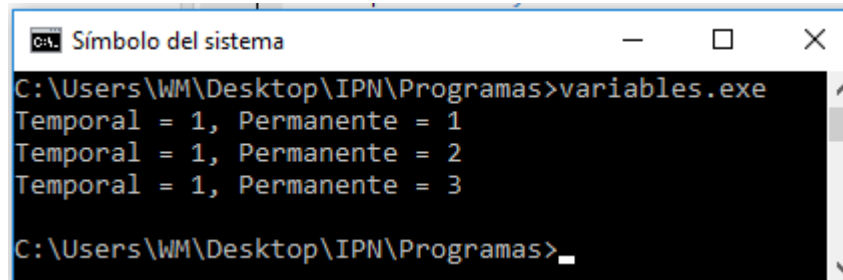
Temporal: El valor de la variable no se mantiene.

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
    int i;
    for (i = 0; i < 3; ++i) {
        int temporal = 1;
        static int permanente = 1;
        printf("Temporal = %d, Permanente = %d\n", temporal, permanente);
        ++temporal;
        ++permanente;
    }
    return 0;
}
```

Almacenamiento temporal.

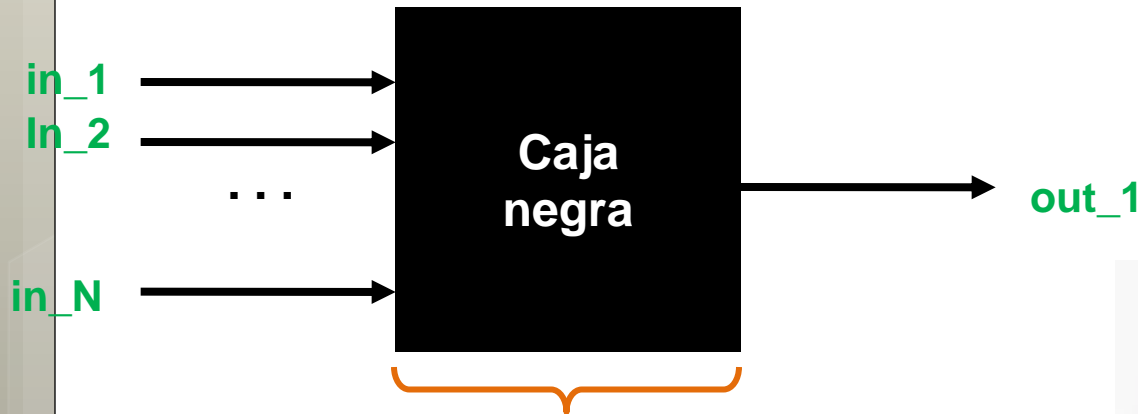
Almacenamiento permanente



```
Símbolo del sistema
C:\Users\WM\Desktop\IPN\Programas>variables.exe
Temporal = 1, Permanente = 1
Temporal = 1, Permanente = 2
Temporal = 1, Permanente = 3
C:\Users\WM\Desktop\IPN\Programas>_
```

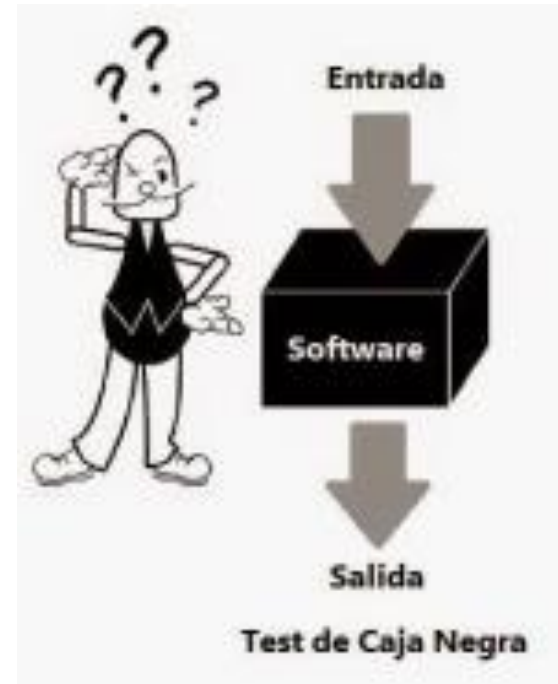

INTRODUCCION A LAS FUNCIONES

Definición: Un función no es mas que un conjunto de instrucciones que realizan cierta tarea.



Se tiene un conocimiento de lo que hace, pero no de cómo lo hace

Centrarse en el funcionamiento del programa.



INTRODUCCION A LAS FUNCIONES



- ¿Que aspectos tengo que tener en cuenta al trabajar con funciones?
- ¿Cómo las uso para algo?
- NO puede ser, ¿Por donde empiezo?

ASPECTOS A TENER EN CUENTA AL TRABAJAR CON FUNCIONES

Cuando se desea trabajar con funciones se deben tener en cuenta 3 aspectos claves:

- Su interfaz (Entradas y salidas).
- Su implementación (¿Cómo es por dentro?, ¿Que es lo que hace?).
- Su invocación (¿Cómo usarla?).



ESTRUCTURA DE UNA FUNCION EN C

Cabecera de la función

```
tipo_retorno nombre_funcion(tipo_1 param_1, . . . , tipo_N param_N)
```

```
{  
    instruccion_1;  
    . . .  
    instruccion_N;  
    return expresion;  
}
```

Cuerpo de la función

Tipo de dato retornado
por la función

Nombre de la función

Cuerpo de la
función

```
float suma(float num1, float num2) {  
    float res;  
    res = num1 + num2;  
    return res;  
}
```

Argumentos de la
función.

Variable retornada por la función

ESTRUCTURA DE UNA FUNCION EN C

Tipo de dato retornado
por la función

Puede ser cualquier tipo de dato ordinario (int, char, float, double, etc), un puntero a cualquier tipo de dato, o un dato tipo struct. Una funcion puede o no retornar un valor.

Nombre de la función

Puede empezar por una letra o el carácter _. Después de este puede contener tantas letras como números o _ se deseen. Si embargo el compilador ignora a partir de cierta cantidad

```
float suma(float num1, float num2) {  
    float res;  
    res = num1 + num2;  
    return res;  
}
```

Argumentos de la
función.

Cada parámetro es separado por coma, los tipos de datos manejados para cada argumento pueden ser de cualquier tipo de datos simples o compuestos.

Variable retornada por la función

DECLARACION DE UNA FUNCION EN C

- La declaración de una función se denomina prototipo. Básicamente, la declaración describe la interfaz de la función. Al declarar una función, básicamente lo que se esta haciendo es especificar la cabecera.

```
float suma(float num1, float num2);  
int max(int x,int y);  
void imprimir_saludo(void);  
void imprimir_arreglo(int a[],int long);  
int comparar_arreglos(int a1[],int a2[]);
```

- Los nombres de los parámetros se suelen omitir.

```
float suma(float,float);  
int max(int,int);  
void imprimir_saludo(void);  
void imprimir_arreglo(int [],int);  
int comparar_arreglos(int [],int []);
```

- La forma general de la declaración de una función es:

```
tipo_retorno nombre_funcion(tipo_1,. . . , tipo_N);
```

DEFINICION DE UNA FUNCION EN C

- La **definición de la función** consiste de la cabecera de la función (declarador) y un bloque de función. La cabecera de la función especifica el nombre de la función, el tipo el valor de retorno, y los tipos y nombres de sus parámetros. Las sentencias del bloque de la función especifica lo que hace la función.

```
float suma(float num1, float num2) {  
    float res;  
    res = num1 + num2;  
    return res;  
}
```

- Declarar una función dentro de otra es algo prohibido
- La forma general de la declaración de una función es:

```
tipo_retorno nombre_funcion(tipo_1 param_1, . . . ,  
tipo_N param_N) {  
    instruccion_1;  
    . . .  
    instruccion_N;  
    return expresion;  
}
```

IMPLEMENTANDO FUNCIONES EN C

- Una función debe ser declarada antes de ser usado, por ello suele ser común colocar las declaraciones antes de la función main.

```
#include <archivo_cabecera>

protipo_funciones;
declaracion_variables_globales;

int main() {
    declaracion_variables_locales;
    funcion_1();
    funcion_2();
    . . .
    funcion_N();
    return 0;
}

funcion_1() {
    codigo_funcion_1;
}

. . .
funcion_N() {
    codigo_funcion_N;
}
```

```
#include <archivo_cabecera>

float suma(float, float);

int main() {
    . . .
    return 0;
}

float suma(float num1, float num2)
{
    float res;
    res = num1 + num2;
    return res;
}
```


INVOCACION DE FUNCIONES EN C

- Una vez que se han implementado las funciones el siguiente paso consiste en usarlas (es decir invocarlas).
- Cuando se ejecuta un programa la primera función que se ejecuta es la función main, por ello su importancia.
- Una función puede invocar otra función.

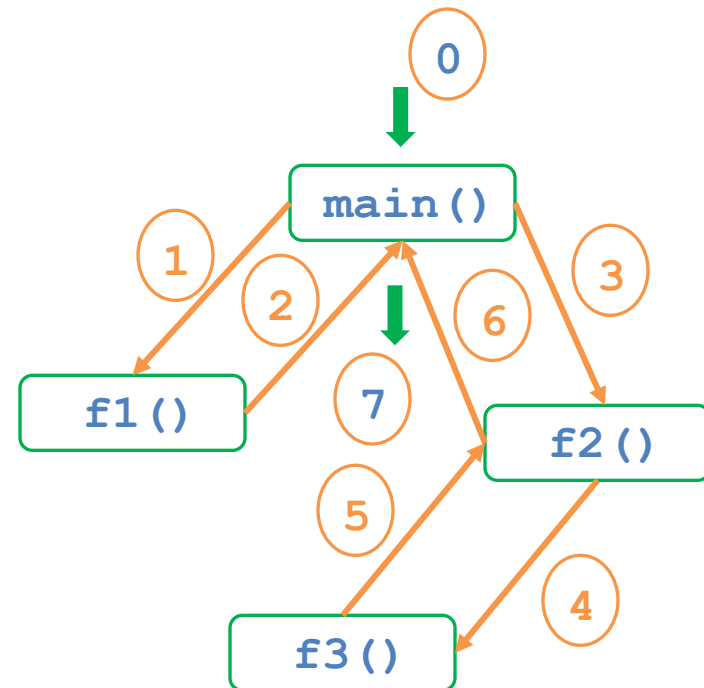
```
#include <stdio.h>

float suma(float, float);

int main() {
    f1();
    f2();
    return 0;
}

f1() {
    . . .
}

f2() {
    f3()
    . . .
}
```



INVOCACION DE FUNCIONES EN C

```
#include <stdio.h>

float suma(float, float);

int main() {
    int a = 3, b = 2, c = 0, d = 3;
    int e, f, g, h, i, k;
    d = suma(2, -4);
    c = suma(a, b);
    f = suma(2, c++);
    g = suma(++c, d++);
    h = suma(suma(1, a), d);
    suma(2, 3);
    return 0;
}

float suma(float num1, float num2) {
    float res;
    res = num1 + num2;
    return res;
}
```

Una función se puede invocar las veces que se desee.

Cuando se invoca un función que retorna un valor se es libre de llevar el valor retornado a una variable.

ANOTACIONES SOBRE FUNCIONES

```
#include <stdio.h>

void saludar(void);
void re_saludar(int);
int ver_algo(void);

int algo = 1;
int main() {
    int c;
    saludar();
    resaludar(3);
    c = ver_algo();
    printf("algo vale: %d", c);
    return 0;
}

void saludar(void) {
    printf("Hola amigo!!\n");
}

void re_saludar(int N){
    int i;
    for(i=0;i<N;i++) {
        printf("Hola !!\n");
    }
}

int ver_algo(void) {
    return algo;
}
```

Para indicar no retorno de datos se usa la palabra clave **void**, hay tres casos básicamente:

1. Cuando una función no tiene argumentos de entrada ni retorna nada.
2. Cuando una función tiene argumentos de entrada pero no devuelve nada.
3. Cuando una función no pide argumentos de entrada pero devuelve algo.

ANOTACIONES SOBRE FUNCIONES EN C

```
#include <stdio.h>

double media(double, double);
int comparar(int, int);

int main() {
    int c;
    double m = 2.5;
    c = comparar(2, 3);
    m = media(m, 3, 3);
    media(3.2, 3.3);
    return 0;
}

double media(double a, double b) {
    double m;
    m = (a+b)/2;
    return m;
}

int comparar(int a, int b) {
    if(a>=b) {
        return a;
    }
    else {
        return b;
    }
}
```

- Una función puede devolver cualquier tipo de dato simple (char, short, int, long, etc), compuesto (estructura) o un puntero a cualquiera de estos dos tipos.
- Una función no puede retornar un array o una matriz.
- Una función solo puede retornar un solo valor a menos que devuelva un puntero o una estructura.

ANOTACIONES SOBRE FUNCIONES EN C

```
#include <stdio.h>

void sumar_array(int [],int [], int);
void restar_array(int *,int *, int);
int main() {
    int a[] = {2,3,-1,4};
    int b[4] = {2,3,-1,4};
    c[4]={0,0,0,0};
    sumar_array(a,b,4);
    sumar_array(c,b,4);
    return 0;
}

void sumar_array(int a[],int b[], int len)
{
    int i;
    for(i=0;i<len;i++) {
        b[i] = b[i] + a[i];
    }
}

void restar_array(int *a,int *b, int len) {
    int i;
    for(i=0;i<len;i++) {
        b[i] = b[i] - a[i];
    }
}
```

- Es posible usar vectores y matrices como parámetros de una función.
- Para declarar un array como argumento se usa la siguiente sintaxis en la definición:
tipo nombre_parametro[],
*tipo *nombre_parametro.*

ANOTACIONES SOBRE FUNCIONES EN C

- También es posible usar matrices como parámetros en una función.

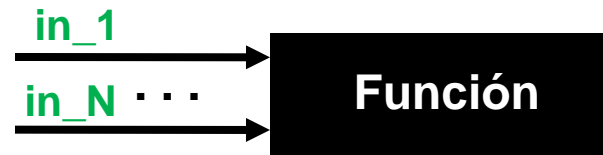
```
double maximum( int , int , double [][] )  
  
. . .  
  
double maximum( int nrows, int ncols, double matrix[nrows][ncols] )  
{  
    double max = matrix[0][0];  
    int c,r;  
    for ( r = 0; r < nrows; ++r )  
        for ( c = 0; c < ncols; ++c )  
            if ( max < matrix[r][c] )  
                max = matrix[r][c];  
    return max;  
}
```

INTERFAZ DE UNA FUNCION

Consiste en definir cuales van a ser las entradas y las salidas de la función, básicamente encontramos 4 tipos.



Funciones sin
entradas ni salidas



Funciones con entradas
pero sin salidas



Funciones con salidas
pero sin entradas



Funciones con entradas y
salidas

INTERFAZ DE UNA FUNCION

Función

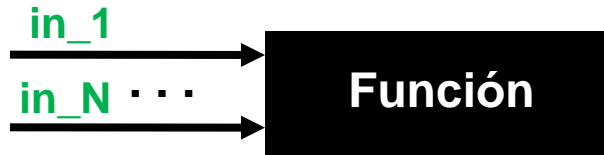
Funciones sin
entradas ni salidas

Ejemplo: Realizar una función que imprima: Hola, buen dia.

saludar

`Hola,buen dia`

INTERFAZ DE UNA FUNCION



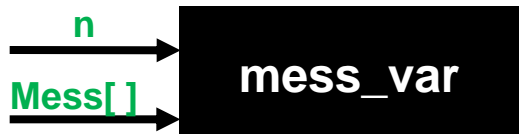
Funciones con entradas
pero sin salidas

Realizar una función que imprima:
saludo las veces que el usuario lo solicite



Digite el numero de saludos: 3
Hola,buen dia
Hola,buen dia
Hola,buen dia

Realizar una función que imprima
lo que el usuario quiera las veces
que este lo solicite



Digite un mensaje: Hola
Digita cuantas veces se imprimira: 3
Hola
Hola
Hola

INTERFAZ DE UNA FUNCION

Función

out_1

**Funciones con salidas
pero sin entradas**

Realizar una función que aumente
una variable.

aumentar

N

Digite el numero: 3
Numero aumentado: 4

INTERFAZ DE UNA FUNCION



Funciones con entradas y salidas

Realizar una función que calcule el factorial de un numero:



Digite el numero: 6
 $6! = 720$

Pasar valores por valor y por referencia

- Por valor: se copia el contenido en otra variable en la función invocada.
- Por referencia: se copia la dirección de la variable.

```
#include<stdio.h>

void sumar(int,int,int,int*);

int main(){
    int v=5,suma;
    sumar(4,v,v*2-1,&suma);
    printf("%d\n",suma);
    return 0;
}

void sumar(int a, int b, int c, int *s){
    b+=2;
    *s = a+b+c;
}
```

Encabezado de la biblioteca estándar

Explicación

<code><assert.h></code>	Contiene macros e información para agregar diagnósticos y ayudar en la depuración de programas.
<code><ctype.h></code>	Contiene los prototipos de las funciones que evalúan ciertas propiedades de los caracteres, prototipos de funciones para convertir letras de minúscula a mayúscula y viceversa.
<code><errno.h></code>	Define macros que son útiles para reportar condiciones de error.
<code><float.h></code>	Contiene los límites del sistema con respecto al tamaño de los números de punto flotante.
<code><limits.h></code>	Contiene los límites del sistema con respecto al tamaño de números enteros.
<code><locale.h></code>	Contiene prototipos de funciones e información adicional que permite modificar un programa para adecuarlo al “local” en el que se ejecuta. La idea de “local” permite al sistema de cómputo manipular diferentes convenciones para expresar datos como fechas, horas, montos en moneda y grandes números alrededor del mundo.
<code><math.h></code>	Contiene los prototipos de las funciones matemáticas de la biblioteca.
<code><setjmp.h></code>	Contiene los prototipos de las funciones que permiten evitar la llamada de función usual y la secuencia de retorno.
<code><signal.h></code>	Contiene prototipos de funciones y macros para manipular varias condiciones que se pudieran presentar durante la ejecución del programa.
<code><stdarg.h></code>	Define macros para tratar con una lista de argumentos correspondientes a una función, cuyos números y tipos son desconocidos.
<code><stddef.h></code>	Contiene definiciones comunes de los tipos utilizados por C para realizar ciertos cálculos.
<code><stdio.h></code>	Contiene los prototipos de las funciones de entrada/salida de la biblioteca estándar, y la información que utilizan.
<code><stdlib.h></code>	Contiene los prototipos de las funciones para la conversión de números a texto y de texto a números, asignación de memoria, números aleatorios, y otras funciones de utilidad.
<code><string.h></code>	Contiene los prototipos de las funciones para el procesamiento de cadenas.
<code><time.h></code>	Contiene prototipos de funciones y tipos para manipular la fecha y la hora.

Matrushka

- La Matrushka es una artesanía tradicional rusa. Es una muñeca de madera que contiene otra muñeca más pequeña dentro de sí. Ésta muñeca, también contiene otra muñeca dentro. Y así, una dentro de otra.



Recursividad: el concepto

- La recursividad es un concepto fundamental en matemáticas y en computación.
- Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.
- Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

Función recursiva

Las funciones recursivas se componen de:

- Caso base: una solución simple para un caso particular (puede haber más de un caso base).

Función recursiva

- Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
 1. El procedimiento se llama a sí mismo
 2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
 3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará

Ejemplo: factorial

Escribe un programa que calcule el factorial (!) de un entero no negativo. He aquí algunos ejemplos de factoriales:

- $1! = 1$
- $2! = 2 \rightarrow 2 * 1$
- $3! = 6 \rightarrow 3 * 2 * 1$
- $4! = 24 \rightarrow 4 * 3 * 2 * 1$
- $5! = 120 \rightarrow 5 * 4 * 3 * 2 * 1$

Ejemplo: factorial (iterativo - repetetitivo)

int factorial (int n)

comienza

fact \leftarrow 1

para i \leftarrow 1 hasta n

fact \leftarrow i * fact

regresa fact

termina

```
int factorial (int n) {  
    int fact = 1;  
    for(int i=1;i<=n;i++)  
        fact = i * fact;  
    return fact;  
}
```

Ejemplo: factorial (recursivo)

int factorial (int n)

comienza

si n = 0 entonces

regresa 1

otro

regresa factorial(n-1)*n

termina

```
int factorial(int n){  
    if(n == 0)  
        return 1;  
    else  
        return factorial(n-1)*n;  
}
```

Ejemplo:

- A continuación se puede ver la secuencia de factoriales.

□ $0! = 1$		
□ $1! = 1$	$= 1 * 1$	$= 1 * 0!$
□ $2! = 2$	$= 2 * 1$	$= 2 * 1!$
□ $3! = 6$	$= 3 * 2 * 1$	$= 3 * 2!$
□ $4! = 24$	$= 4 * 3 * 2 * 1$	$= 4 * 3!$
□ $5! = 120$	$= 5 * 4 * 3 * 2 * 1$	$= 5 * 4!$
□ ...		
□ $N! =$	$= N * (N - 1)!$	

Solución

Aquí podemos ver la secuencia que toma el factorial

$$N! = \begin{cases} 1 & \text{si } N = 0 \text{ (base)} \\ N * (N - 1)! & \text{si } N > 0 \text{ (recursión)} \end{cases}$$

Un razonamiento recursivo tiene dos partes: la base y la regla recursiva de construcción. La base no es recursiva y es el punto tanto de partida como de terminación de la definición.

Solución Recursiva

Dado un entero no negativo x , regresar el factorial de x

fact:

Entrada n entero no negativo,

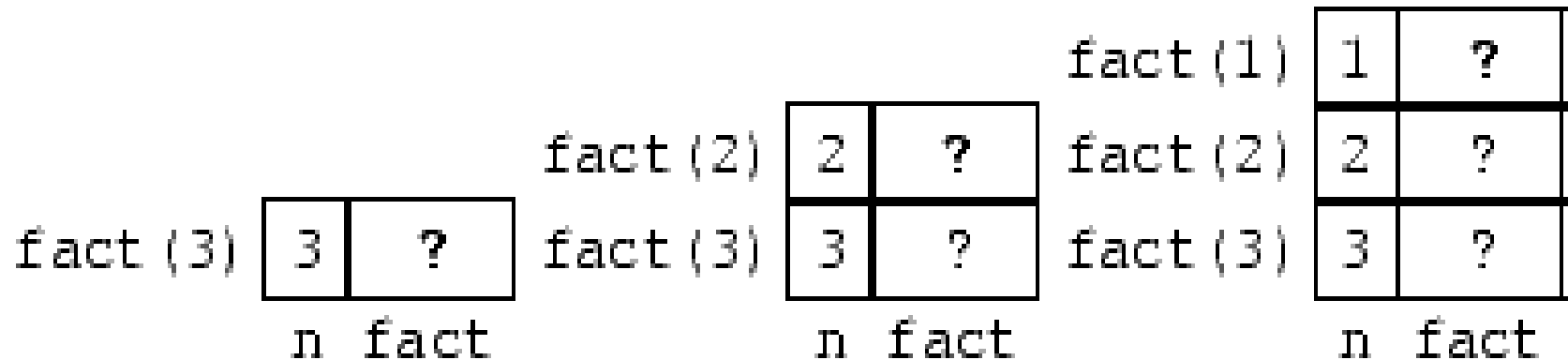
Salida: entero.

```
int fact (int n)
{
    if (n == 0)
        return 1;
    else
        return fact(n - 1) * n ;
}
```

Es importante determinar un caso base, es decir un punto en el cual existe una condición por la cual no se requiera volver a llamar a la misma función.

¿Cómo funciona la recursividad?

Llamadas recursivas



Resultados de las llamadas recursivas

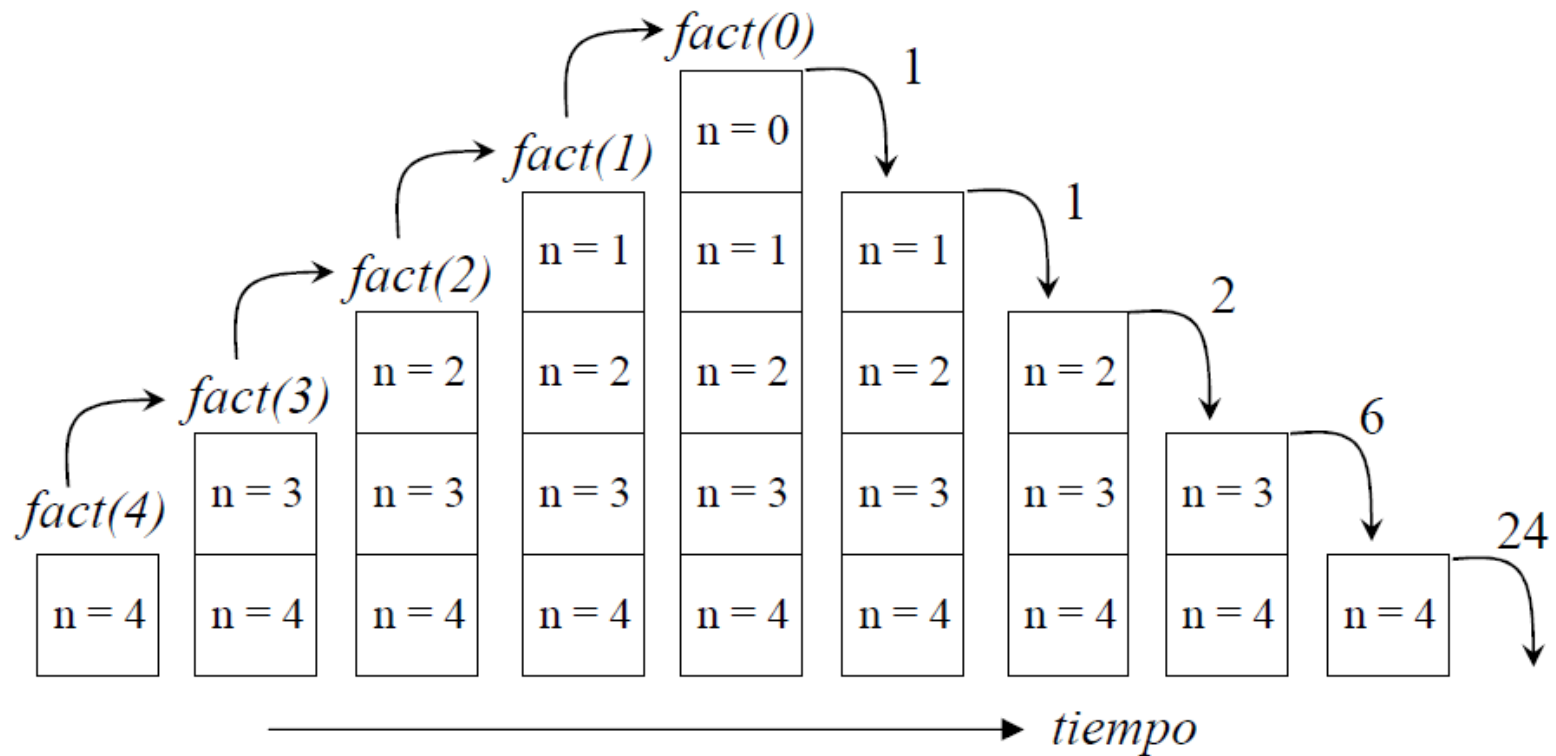
0	1
1	?
2	?
3	?
n fact	

1	1
2	?
3	?
n fact	

2	2
3	?
n fact	

3	6
n fact	

¿Cómo funciona la recursividad?



¿Por qué escribir programas recursivos?

- Son mas cercanos a la descripción matemática.
- Generalmente mas fáciles de analizar
- Se adaptan mejor a las estructuras de datos recursivas.
- Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

Factible de utilizar recursividad

- Para simplificar el código.
- Cuando la estructura de datos es recursiva ejemplo : árboles.

No factible utilizar recursividad

- Cuando los métodos usen arreglos largos.
- Cuando las iteraciones sean la mejor opción.

Otros conceptos

- Cuando un procedimiento incluye una llamada a sí mismo se conoce como **recursión directa**.
- Cuando un procedimiento llama a otro procedimiento y éste causa que el procedimiento original sea invocado, se conoce como **recursión indirecta**.

Ejemplo: Serie de Fibonacci

Valores: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

Cada término de la serie suma los 2 anteriores. Fórmula recursiva

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$$

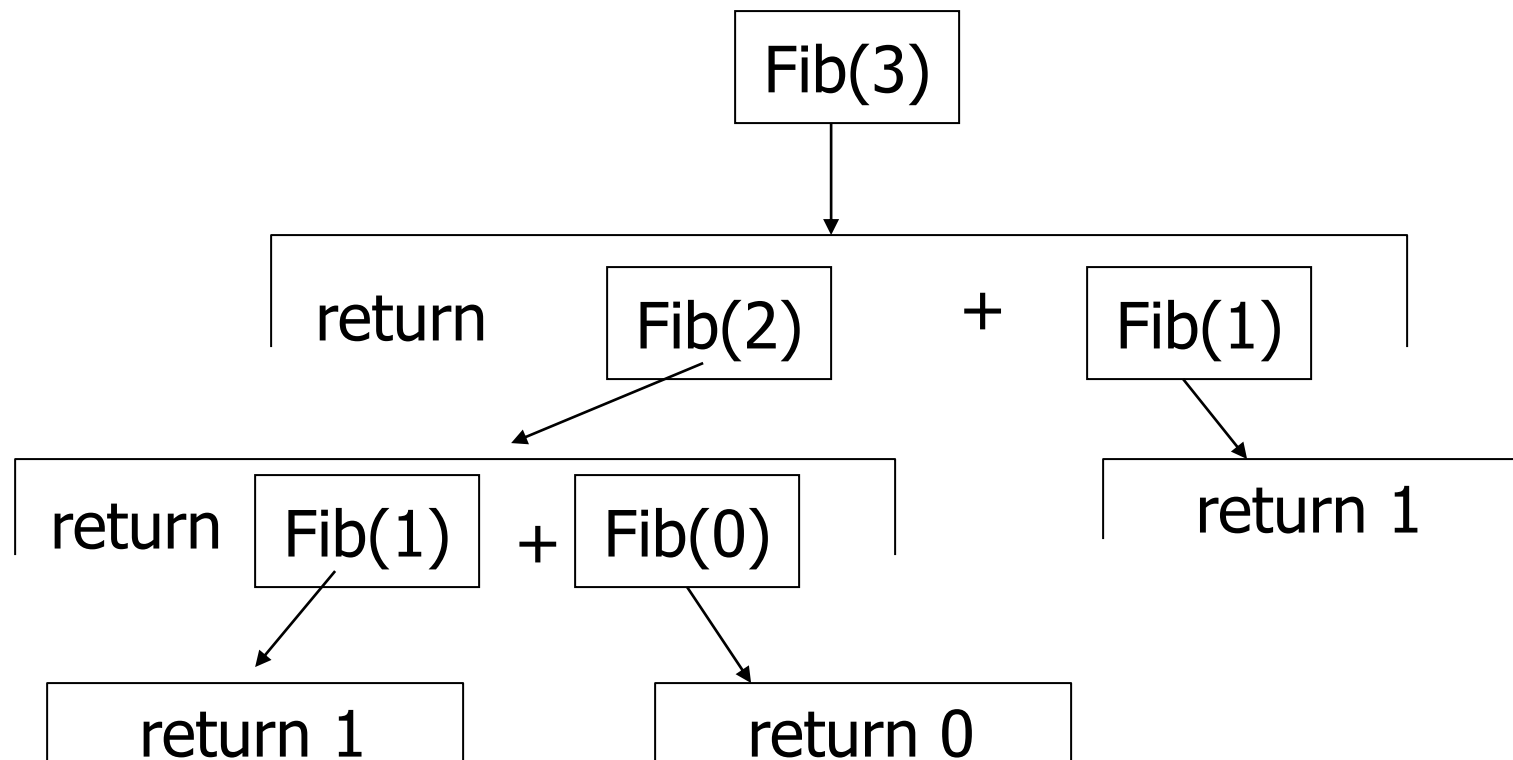
Caso base: Fib (0)=0; Fib (1)=1

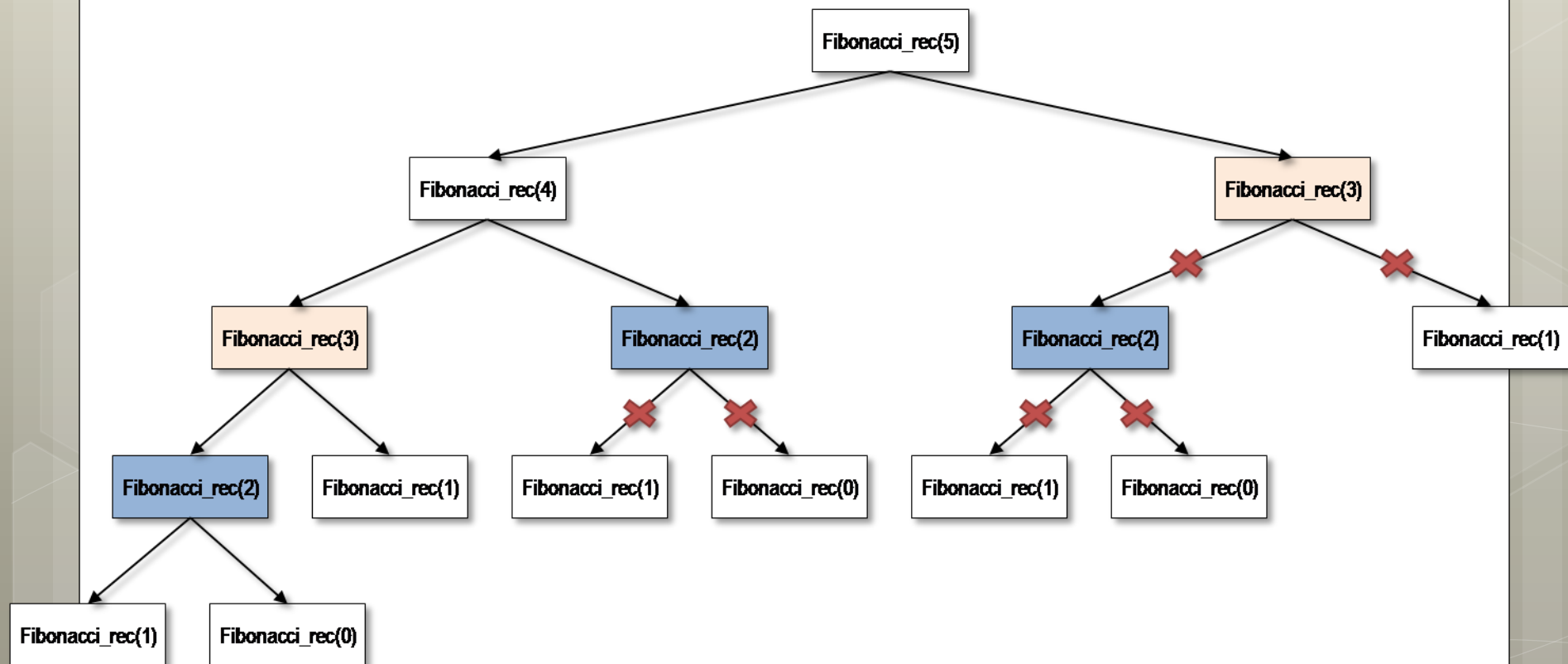
Caso *recursivo*: Fib (i) = Fib (i -1) + Fib(i -2)

```
int fib(int n) {  
    if (n <= 1)  
        return n; //condición base  
    else  
        return fib(n-1)+fib(n-2) ; //condición recursiva  
}
```

Ejemplo: Serie de Fibonacci

Traza del cálculo recursivo





Recursión vs iteración

Repetición

Iteración: ciclo explícito (se expresa claramente)

Recursión: repetidas invocaciones a método

Terminación

Iteración: el ciclo termina o la condición del ciclo falla

Recursión: se reconoce el caso base

En ambos casos podemos tener ciclos infinitos

Considerar que resulta más positivo para cada problema

- LA RECURSIVIDAD SE DEBE USAR CUANDO SEA REALMENTE NECESARIA, ES DECIR, CUANDO NO EXISTA UNA SOLUCIÓN ITERATIVA SIMPLE.

Dividir para vencer

- Muchas veces es posible dividir un problema en subproblemas más pequeños, generalmente del mismo tamaño, resolver los subproblemas y entonces combinar sus soluciones para obtener la solución del problema original.
- Dividir para vencer es una técnica natural para las estructuras de datos, ya que por definición están compuestas por piezas. Cuando una estructura de tamaño finito se divide, las últimas piezas ya no podrán ser divididas.

<http://conclase.net/c/librerias>

FUNCIONES DE CARACTER

El archivo cabecera [`<ctype.h>`](#) define un grupo de funciones y macros para la manipulación de variables tipo carácter. La siguiente tabla resume las principales funciones:

Función	Descripción
int <code>isalpha</code> (int c)	Testea si el carácter es alfanumérico
int <code>isdigit</code> (int c)	Testea si el carácter es un dígito entre 0 y 9
int <code>isupper</code> (int c)	Testea si el carácter introducido es una letra mayúscula (A - Z).
int <code>islower</code> (int c)	Testea si el carácter introducido es una letra mayúscula (a - z).
int <code>isalnum</code> (int c)	Evalúa si el carácter introducido es una letra o dígito.
int <code>isctrl</code> (int c)	Testea si el carácter introducido es de control.
int <code>isxdigit</code> (int c)	Testea si el carácter es un dígito hexadecimal
int <code>isprint</code> (int c)	Test de carácter imprimible incluyendo espacio.
int <code>isgraph</code> (int c)	Test de carácter imprimible excepto espacio.
int <code>isspace</code> (int c)	Test de carácter de espacio (espacio, avance de página, nueva línea, retorno de carro, tabulación, tabulación vertical)
int <code>ispunct</code> (int c)	Testea por carácter de puntuación.
int <code>toupper</code> (int c)	Convierte a letras mayúsculas.
int <code>tolower</code> (int c)	Convierte a letras minúsculas.

FUNCIONES DE ENTRADA SALIDA

El archivo cabecera [<stdio.h>](#) contiene funciones de utilidad para la entrada y salida de datos.

Entrada (desde teclado) y salida (hacia pantalla) formateada	
Función	Descripción
<u>scanf</u>	Lee una cadena formateada desde stdin (teclado).
<u>printf</u>	Imprime una cadena formateada a stdout (pantalla).
<u>getchar</u>	Obtiene un carácter desde stdin (teclado).
<u>putchar</u>	Escribe un carácter a stdout (pantalla).
<u>gets</u>	Obtiene una cadena de caracteres desde stdin (teclado).
<u>puts</u>	Escribe un carácter a stdout (pantalla).

FUNCIONES DE LA LIBRERÍA STDLIB

El archivo cabecera [<stdlib.h>](#) contiene funciones para manejo dinámico de memoria, generación de números aleatorios, comunicación con el ambiente, aritmética entera, búsqueda, ordenamiento y conversión.

Entrada (desde teclado) y salida (hacia pantalla) formateada	
Función	Descripción
<u>atof</u>	Convierte un string a double.
<u>atoi</u>	Convierte un string a entero.
<u>rand</u>	Genera un numero aleatorio.
<u>srand</u>	Inicializa un generador de números aleatorios.

FUNCIONES CON CADENAS DE CARACTERES

El archivo cabecera [<string.h>](#) contiene la mayoría de las funciones empleadas para la manipulación de cadenas de caracteres

Funciones para copiar	
Función	Descripción
<u>strcpy</u>	Copia un string.
<u>strncpy</u>	Copia un numero determinado de caracteres de un string.
Funciones para concatenar	
Función	Descripción
<u>strcat</u>	Concatena strings.
<u>strncat</u>	Añade un numero determinado de caracteres a un string.
Funciones para comparar	
Función	Descripción
<u>strcmp</u>	Compara dos cadenas de caracteres.
<u>strncmp</u>	Compara un determinado numero de caracteres entre dos cadenas de caracteres.
Funciones para buscar	
Función	Descripción
<u>strpbrk</u>	Localiza un carácter en un string.
<u>strtok</u>	Divide un string en tokens.
<u>strcspn</u>	Revisa una primera cadena de caracteres hasta que encuentra la primera ocurrencia de uno de los caracteres obtenidos la otra cadena.
<u>strchr</u>	Localiza la primera ocurrencia de un string en una cadena de caracteres
<u>strrchr</u>	Localiza la ultima ocurrencia de un carácter en un string
<u>strstr</u>	Localiza un substring.