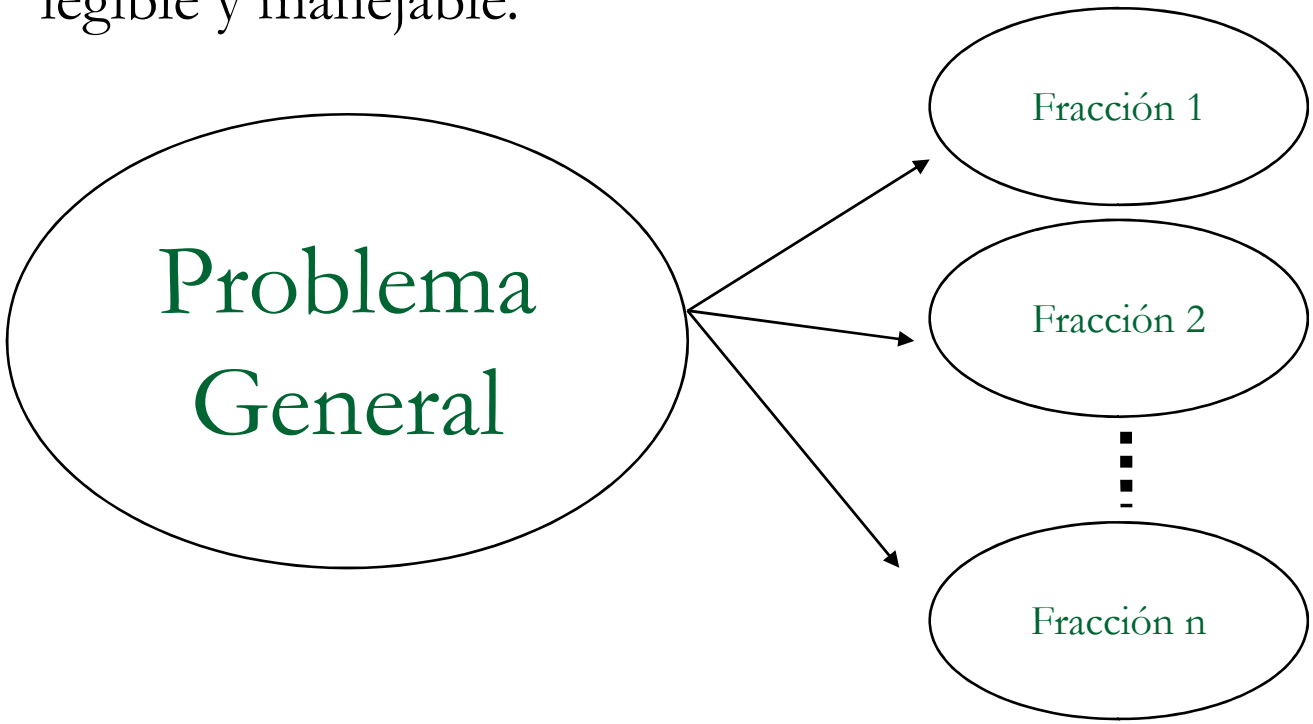
Programación 1

Subprogramas

Prof. Lisbeth Pérez Rivas
lisbethpe@ula.ve

Programación Modular

Paradigma de la programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.



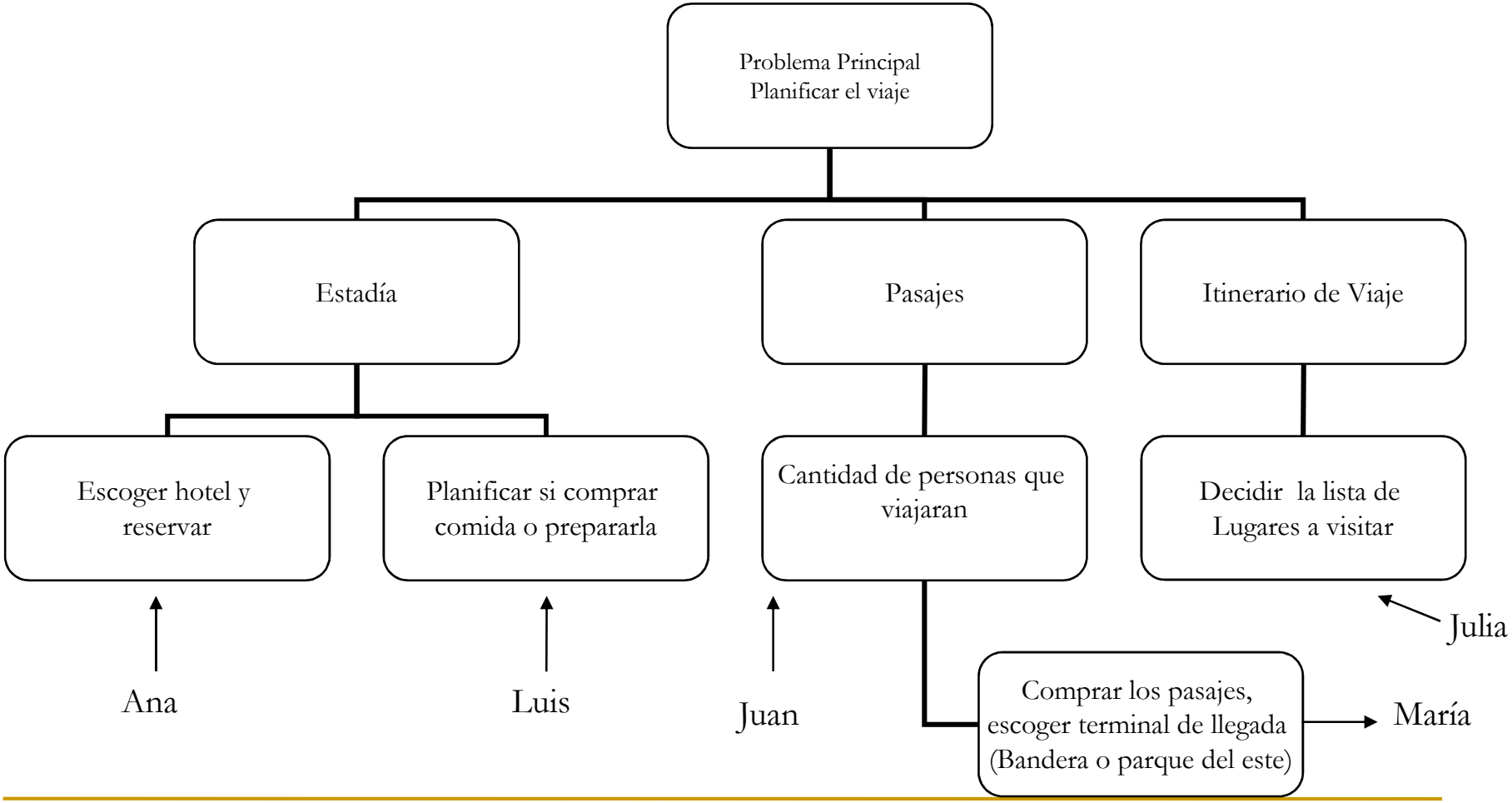
Diseño Descendente

Un problema se divide en problema más pequeños (sub-problemas) los cuales a su vez pueden dividirse en sub-problemas aún más pequeños. Este proceso se realiza hasta que los sub-problemas son tan sencillos que pueden ser resueltos fácilmente.

Esto también es conocido como la técnica de “divide y vencerás”

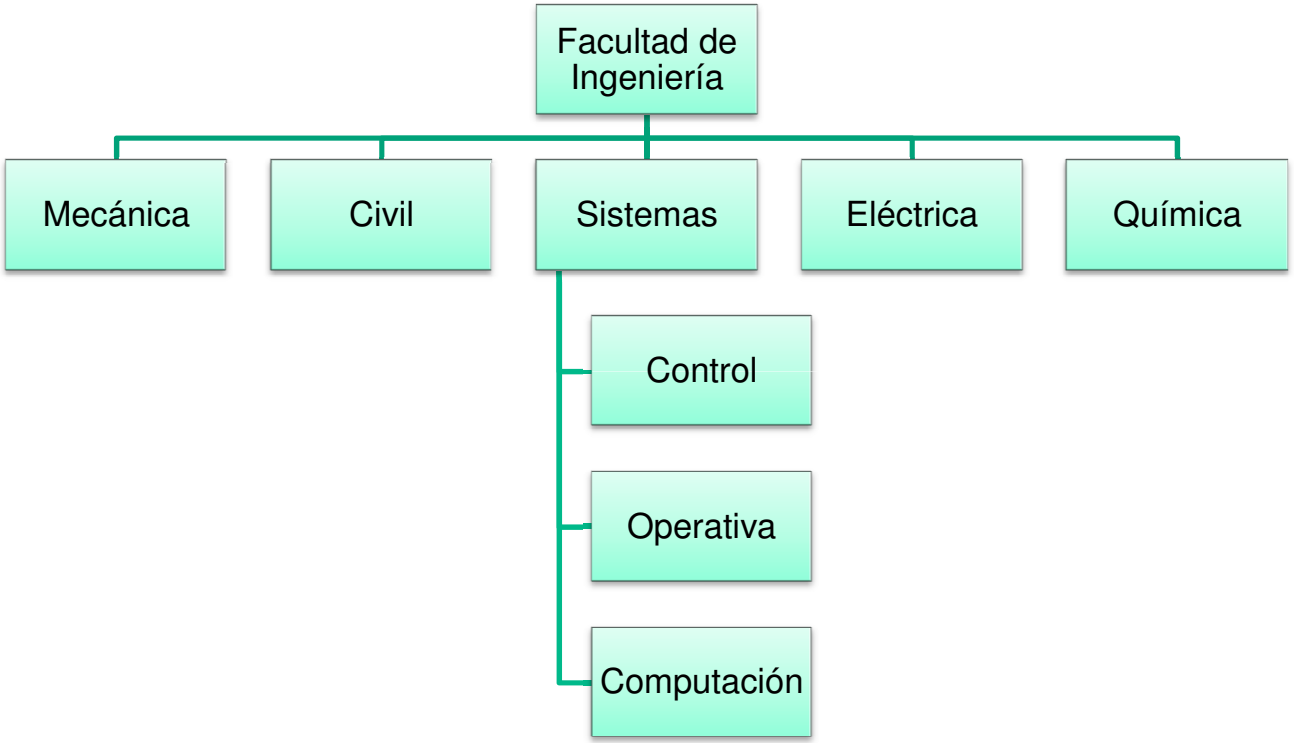
Programación Modular

Planificar un viaje a Caracas



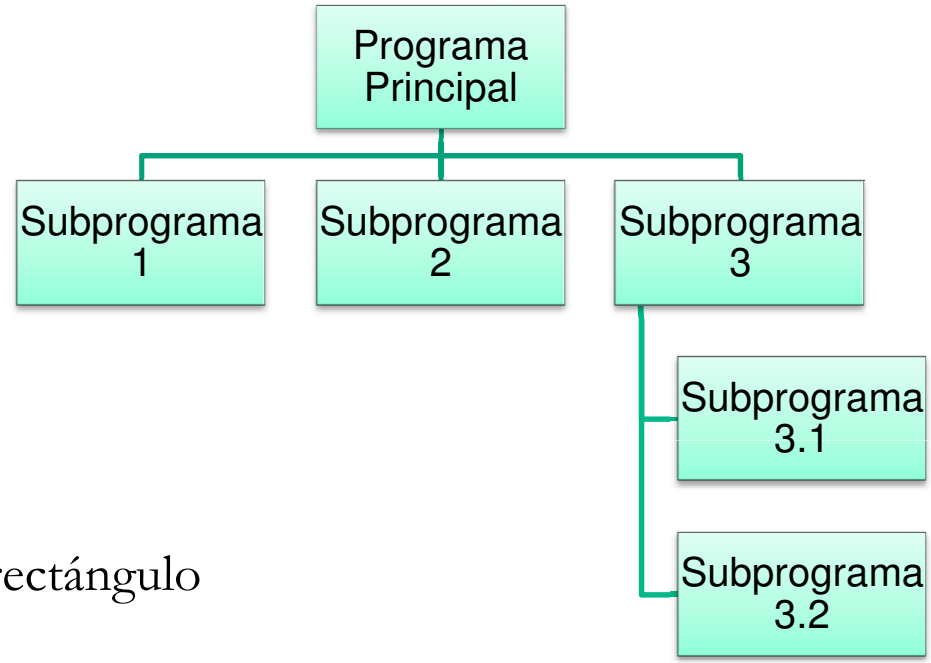
Programación Modular

Llevar el control de la facultad de Ingeniería

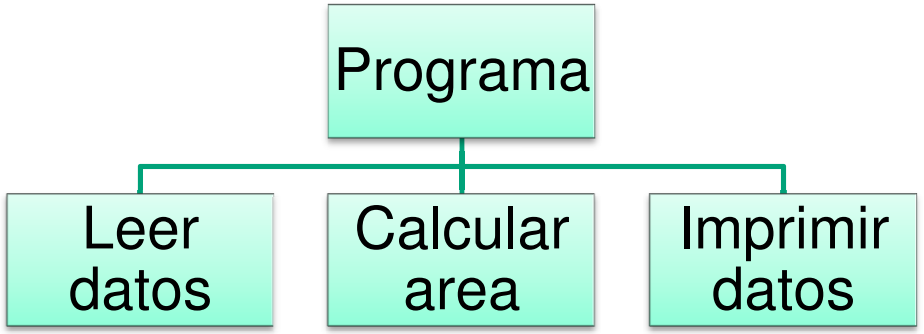


Programación Modular

En términos de programación

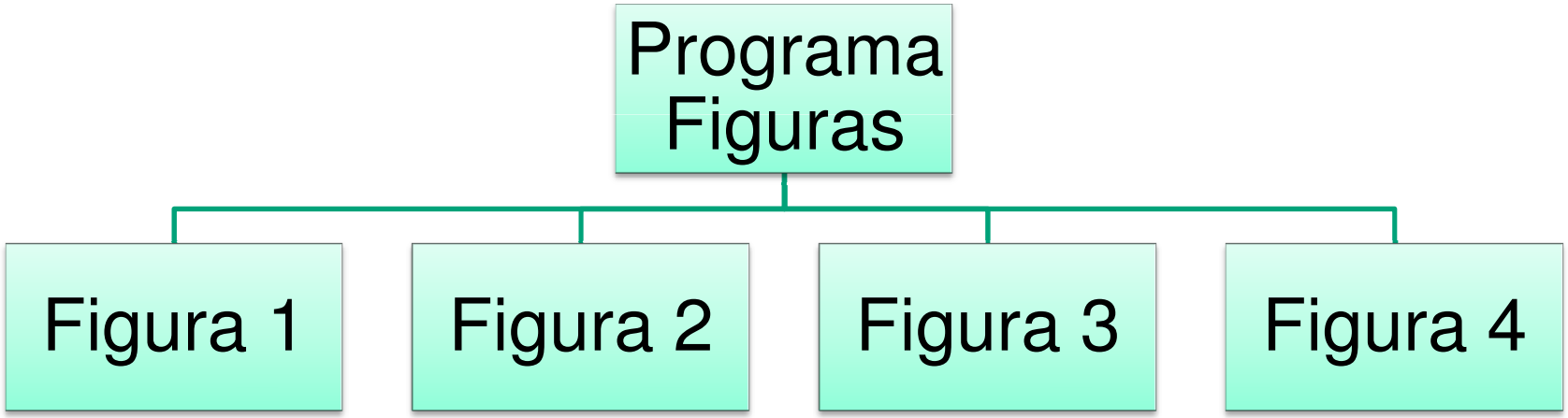


Calcular el área de un rectángulo



Programación Modular

Con el ejercicio de las figuras con asteriscos.



Programación Modular

Ventajas:

- Permite la localización rápida de errores.
- Como existen módulos independientes, varios programadores pueden trabajar simultáneamente y de manera independiente en la resolución de un problema.
- Se puede modificar un módulo sin afectar a los demás.
- Reutilización de módulos.
- Fácil comprensión del programa completo.

Programación Modular

Subprograma

- Procedimientos: Grupo de sentencias que realiza una tarea particular y devuelve cero valores a través del nombre del procedimiento.

- Funciones: Subprograma que devuelve un único resultado al programa principal o subprograma que los llamó a través del nombre de la función.

Funciones

Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un valor denominado resultado. Por ejemplo:

$$f(x) = \frac{x}{1+x^2}$$

Donde f es el nombre de la función y x es el argumento. Para evaluar f debemos darle un valor actual a x; por ejemplo, con x=3 se obtiene el valor 0.3

$$f(3) = \frac{3}{1+3^2}$$

$$f(3) = \frac{3}{1+9} = \frac{3}{10} = 0.3$$

Funciones

Una función puede tener varios argumentos.

$$f(x,y)=\frac{x-y}{\sqrt{x}-\sqrt{y}}$$

Sin embargo, un único valor se asocia con la función para cualquier par de valores dados a los argumentos

$$f(9,4)=\frac{9-4}{\sqrt{9}-\sqrt{4}}$$

$$f(9,4)=\frac{5}{3-2}=\frac{5}{1}=5$$

Funciones

Una función es diseñada para realizar una tarea específica:

- Tomar una lista de valores actuales.
- Ejecutar sus instrucciones.
- Devolver un **único** valor.

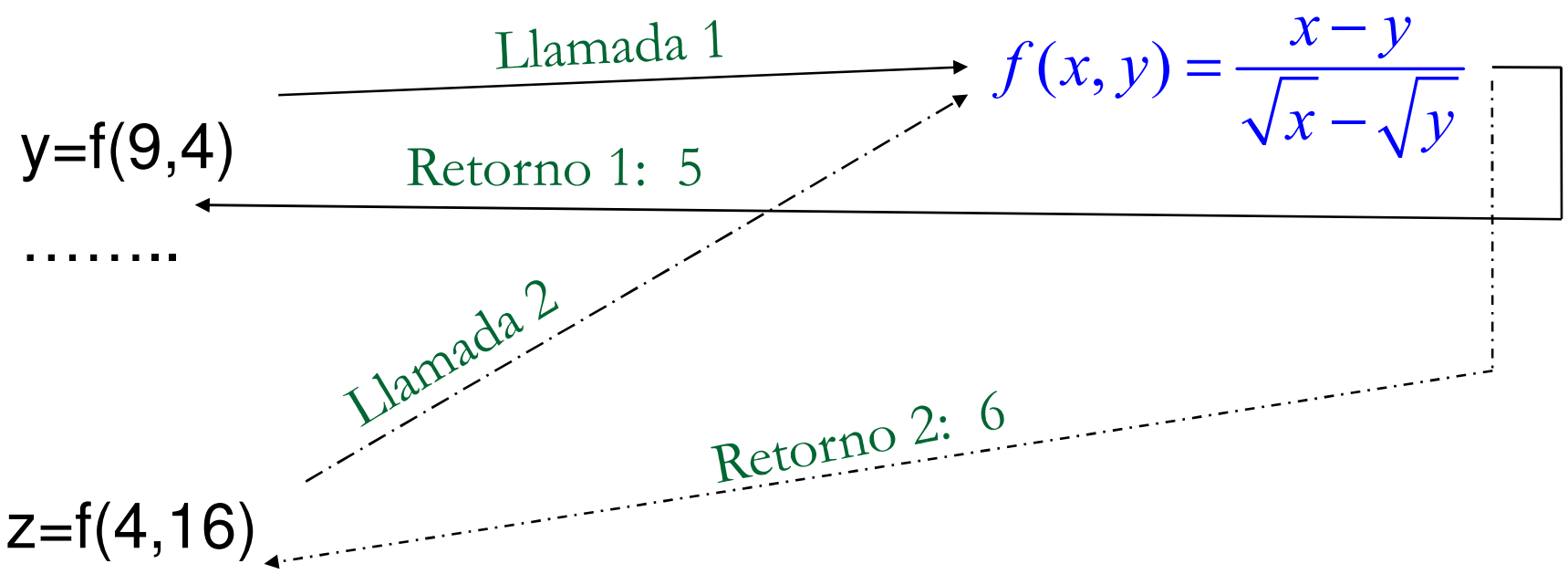
En C, el programa invoca a la función con su nombre seguida por una lista de parámetros actuales que deben coincidir en cantidad, tipo y orden con los de la función que fue definida.

La función es definida una sola vez y puede ser utilizada N veces.

Mecanismo de llamadas

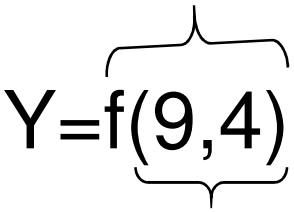
Llamadas a la función

Definición de la función f(x,y):



Qué se necesita para llamar a una función?

El valor de retorno es asignado a la variable Y



Parámetros actuales

Parámetros

- Parámetros Actuales: Valores de llamada de la función y que son tomados por los parámetros formales.

- Parámetros Formales: Declarados en la definición de la función. Ejemplo: f(x,y), x y y son los parámetros formales.

Calcular la raíz cuadrada de un número

```
int main(){
        float numero,resultado;
        printf("inserte numero");
        scanf("%f",&numero);
        resultado=sqrt(numero);
        printf("la raiz cuadrada es: %f",resultado);
        return 0;
}
```

Funciones de bibliotecas: stdio.h

Función	Descripción
Printf	Escritura de datos
Scanf	Lectura de datos
getchar	Lectura de carácter
Operaciones sobre archivos	

Funciones de bibliotecas: math.h

Función	Descripción
Acos	arco coseno
Pow(x,y)	Devuelve el valor de x elevado a la potencia y
abs	Valor absoluto
Log	Logaritmo natural
Log10	Logaritmo base 10
Exp	Función Exponencial
Cos	coseno

Funciones definidas por el usuario

Algoritmo de la Declaración:

tipo_resultado **función** nombre _ función (lista_parámetros_formales)

Declaración de variables locales

Inicio

Conjunto de sentencias

Devolver expresión

Fin_funcion

Donde:

tipo_resultado es el tipo de dato que devuelve la función

lista_parametros_formales es la lista de los parámetros que recibe la función.

Funciones definidas por el usuario

La lista de parámetros formales viene dada por:

Tipo_dato nombre_parametro1, tipo_dato nombre_parametro2…

Los nombres de los parámetros solo son validos dentro de la función que son definidos.

Si no son necesarios parámetros la lista queda vacia:

tipo_resultado **función** nombre_función( )

Funciones definidas por el usuario

```
real función f_x(real x, real y)
        real resultado
        inicio
                resultado= (x-y)/(raiz(x)-raiz(y))
                devolver resultado
        fin_funcion
```

Donde:

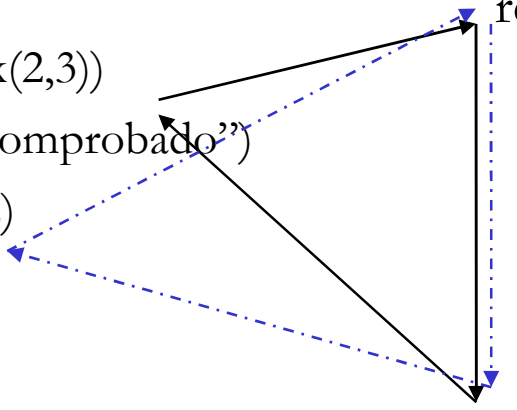
Devolver indica el valor de retorno de la función.

La variable o expresión asociada debe ser del mismo tipo que la función.

Con la sentencia devolver se regresa el flujo del programa al lugar de donde fue invocada la función.

Ejemplo Completo Función f(x,y)

1.Inicio
2.Escribir(f_x(2,3))
3.Escribir("Comprobado")
4.Z=f_x(10,2)
5.Fin



```
real función f_x(real x, real y)
    real resultado
    inicio
            resultado= (x-y)/(raiz(x)-raiz(y))
            devolver resultado
    fin_funcion
```

La llamada a la función.

Escribir(f_x(2,3))

f_x= nombre de la función, debe ser el mismo nombre que aparece en la definición.

El número de parámetros actuales debe ser igual en tipo y orden que los parámetros formales.

El valor de retorno debe ser asignado a una variable, escrito por pantalla o utilizado en alguna expresión.

Ejemplo área del triángulo

```
real función area_t(real b,real h)
        Inicio
                Devolver b*h
        Fin_función
```

1.Inicio
2.Escribir(“inserte la base”)
3.Leer(base)
4.Escribir(“inserte la altura”)
5.Leer(altura)
6.A=area_t(base,altura)
7.Escribir(“el area es : ”,A)
8.Fin

Ejemplo factorial de un número

```
entero función factorial(entero n)
        entero fact, i
        Inicio
                fact=1
                Para (i=1;i≤n;i=i+1)
                        fact=fact*i
                Fin_RP
                Devolver fact
        Fin_función
```

1.Inicio
2.Escribir(“inserte un entero”)
3.Leer(x)
4.z=factorial(x)
5.Escribir(“el factorial de”,x, “es: ”,z)
6.Fin

Definición de una función en C

Tipo_dato nombre_función(lista parametros)

En notación algorítmica	En C
entero función factorial(entero n)	Int factorial(int n)
real función area_t(real b,real h)	float area_t(float b,float h)
entero función principal( )	int main( )

Definición de una función en C

real función area_t(real b,real h)
Inicio
Devolver b*h
Fin_función

1.Inicio
2.Escribir(“inserte la base”)
3.Leer(base)
4.Escribir(“inserte la altura”)
5.Leer(altura)
6.A=area_t(base,altura)
7.Escribir(“el area es : ”,A)
8.Fin

```
Float area_t(float b,float h){
    return (b*h);
}
Int main( ){
    float base,altura,A;
    printf(“inserte la base”);
    scanf(“%f”,&base);
    printf(“inserte la altura”);
    scanf(“%f”,&altura);
    A=area_t(base,altura);
    printf(“el area es: %f”,A);
    return 0;
}
```

Definición de una función en C

```
entero función factorial(entero n)
        entero fact, i
        Inicio
                fact=1
                Para (i=1;i≤n;i=i+1)
                        fact=fact*i
                Fin_RP
                Devolver fact
        Fin_función
```

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

Llamado de la función factorial

1.Inicio
2.Escribir(“inserte un entero”)
3.Leer(x)
4.z=factorial(x)
5.Escribir(“el factorial de”,x, “es: ”,z)
6.Fin

```
#include<stdio.h>
/*aquí va la definición de la función*/
Int main() {
    int x, factorial, z;
    printf(“inserte un entero”);
    scanf(“%d”,&x);
    z=factorial(x);
    printf(“el factorial de %d es:
%d”          ,x,z);
    return 0;
}
```

El programa completo queda como…

```
#include<stdio.h>
Int factorial (int n){
      int fact, i;
      fact=1
      for (i=1;i≤n;i++)
                  fact*=i;
      return (fact);
}
Int main() {
      int x, z;
      printf("inserte un entero");
      scanf("%d",&x);
      z=factorial(x);
      printf("el factorial de %d es:
%d"               ,x,z);
      return 0;
}
```

Que variables están presentes?

En la función: n, i, fact.

En el programa principal:

x, z.

Que pasa si…

```
#include<stdio.h>
Int factorial (int n){
      int fact, i;
      fact=1
      for (i=1;i≤n;i++)
                  fact*=i;
      return (fact);
}
Int main() {
      int x, z;
      printf("inserte un entero");
      scanf("%d",&x);
      z=factorial(x);
      printf("el factorial de %d es:
%d"               ,x,z);
      i++;
      return 0;
}
```

Insertamos la orden i++ en el programa principal?

Se produce un error

Ámbito de las variables

- Variables Locales

- Variables Globales

Variables

- **Locales:** Una variable local es una variable que está declarada dentro de un subprograma, y se dice que es local al subprograma. Y lo que la caracteriza es que su valor sólo está disponible mientras se ejecuta el subprograma.

```
#include<stdio.h>

int suma (int a, int b){
        int r;
        r=a+b;
        return (r);
}
int main( ){
        int n1,n2,s;
        n1=10;
        n2=3;
        s=suma(n1,n2);
        return 0;
}
```

r es una variable local de la función suma y solo está disponible en este ámbito

Variables

```
#include<stdio.h>
int factorial (int n){
      int fact, i;
      fact=1;
      for (i=1;i<=n;i++)
                  fact*=i;
      return (fact);
}
int main() {
      int x, z;
      printf("inserte un entero");
      scanf("%d",&x);
      z=factorial(x);
      printf("el factorial de %d es: %d",x,z);
      return 0;
}
```

Variables locales

De la función factorial: n, i, fact.
Del programa principal: x, z.

Que pasaba al insertar i++ en el principal?

```
#include<stdio.h>
int factorial (int n){
      int fact, i;
      fact=1;
      for (i=1;i<=n;i++)
                  fact*=i;
      return (fact);
}
int main() {
      int x, z;
      printf("inserte un entero");
      scanf("%d",&x);
      z=factorial(x);
      printf("el factorial de %d es:
%d"               ,x,z);
      i++;
      return 0;
}
```

La variable i no existe en el programa principal, pues es local a la función factorial. Por ello solo se puede modificar dentro de la función.

Variables

- **Globales:** Una variable global es una variable que está declarada fuera de las funciones y puede ser utilizada por cualquier función o por el programa principal.

```
#include<stdio.h>
int a,b;
int suma (){
      int r;
      r=a+b;
      return (r);
}
int main( ){
      int s;
      a=10;
      b=3;
      s=suma(a,b);
      return 0;
}
```

a y b son variables globales

s continúa siendo una variable local del programa principal.

Variables

```
#include<stdio.h>
int i;

int suma( ){
        int s;
        s=5+10;
        i=5;
        return s;
}
Int main(){
     i=10;
    printf("i: %d",i);
        printf("El resultado es %d",suma( ));
        printf("i: %d",i);
        return 0;
}
```

Variable Lugar	i	s	Salida
Programa Principal	10	-	10
Función Suma	5	15	15
Programa Principal	5	-	5

Variables

```
#include<stdio.h>


int suma( ){
            int s;
            s=5+10;
            return s;
}
Int main(){
     int i=10;
    printf("i: %d",i);
            printf("El resultado es %d",suma( ));
            printf("i: %d",i);
            return 0;
}
```

Variable Lugar	i	s	Salida
Programa Principal	10	-	10
Función Suma	5	15	15
Programa Principal	10	-	10

Variables

```
#include<stdio.h>


int suma( ){
            int s,i;
            s=5+10;
            i=20;
            return s;
}
Int main(){
     int i=10;
    printf("i: %d",i);
            printf("El resultado es %d",suma( ));
            printf("i: %d",i);
            return 0;
}
```

Variable Lugar	i	s	Salida
Programa Principal	10	-	10
Función Suma	20	15	15
Programa Principal	10	-	10

Procedimientos

Grupo de sentencias que realiza una tarea particular y devuelve cero valores a través del nombre del procedimiento. Para ello se utiliza el tipo VOID

En Código:

```
void suma( int a, int b){
    int s;
    s=a+b;
    printf("%d",s);
}
```

En algoritmo:

```
vacio suma( entero a, entero b)
   entero s
   Inicio
s=a+b
Escribir (s)
   Fin_funcion
```

Procedimientos

Mostrar una pantalla como se presenta a continuación y leer la opción que desee el usuario

Figuras

Tipo de figura:
1. triángulo
2. triángulo invertido
0. Salir

Inserte una opción: _____

Para hacer esto podemos:

1. Crear la pantalla en el programa principal y leer las opciones.
2. Crear un procedimiento que muestra la pantalla y en el programa principal leer las opciones
3. Crear un procedimiento que muestre la pantalla, una función que lea la opción, e invocarlos desde el programa principal.

Procedimientos

```
Void mostrar_pantalla( ){
printf(“\n\t\tFiguras\n”);
printf(“\n1. Triangulo”);
printf(“\n2. Triangulo invertido”);
printf(“\n0. Salir”);
printf (“\n\nInserte una opcion: ___”);
}
```

```
Int leer_opcion( ){
int opcion;
scanf(“%d”,&opcion);
return (opcion);
}
```

```
Int main( ){
int op;
mostrar_pantalla();
op=leer_opcion();
return 0;
}
```

Como ordenar el programa completo?

Opción 1:

- Declaración de librerías.
- Definición de constantes y variables globales.
- Definición de funciones.
- Programa principal.

Como ordenar el programa completo?

Opción 1:

```
#include<stdio.h>
Void mostrar_pantalla( ){
printf(“\n\t\tFiguras\n”);
printf(“\n1. Triangulo”);
printf(“\n2. Triangulo invertido”);
printf(“\n0. Salir”);
printf (“\n\nInserte una opcion: ___”);
}
Int leer_opcion( ){
int opcion;
scanf(“%d”,&opcion);
return (opcion);
}
Int main( ){
int op;
mostrar_pantalla();
op=leer_opcion();
return 0;
}
```

Como ordenar el programa completo?

Opción 2:

- Declaración de librerías.
- Definición de constantes y variables globales.
- **Definición de prototipos.**
- Programa principal.
- Definición de funciones.

Prototipos de funciones

Cuando las funciones no se definen antes de la función principal, es necesario escribir los prototipos de las mismas. Esto es:

Tipo_retorno nombre_funcion (tipos de datos de los parametros);

Prototipos de funciones

Ejemplos:

```
Int leer_opcion( );
void mostrar_pantalla( );
int suma(int,int);
float calculo(float,float,float);
```

Como ordenar el programa completo?

```
#include<stdio.h>
Void mostrar_pantalla( );
Int leer_opcion( );
 Int main( ){
 int op;
 mostrar_pantalla();
 op=leer_opcion();
 return 0;
 }
 Void mostrar_pantalla( ){
 printf(“\n\t\tFiguras\n”);
 printf(“\n1. Triangulo”);
 printf(“\n2. Triangulo invertido”);
 printf(“\n0. Salir”);
 printf (“\n\nInserte una opcion: ___”);
 }
 Int leer_opcion( ){
 int opcion;
 scanf(“%d”,&opcion);
 return (opcion);
 }
```

Paso de parámetros

La definición de una función es:

Tipo_dato_retorno nombre_funcion(lista_parametros)

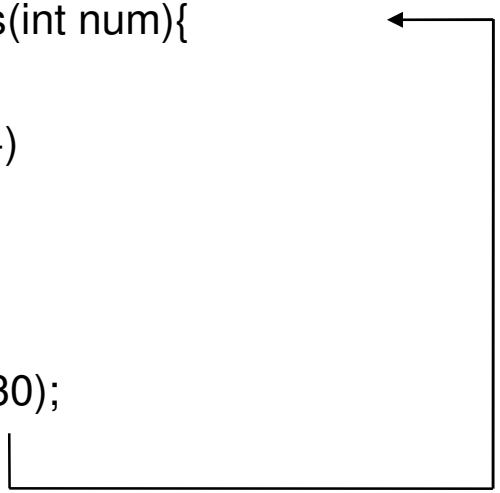
Un parámetro es una variable cuyo valor debe ser: proporcionado por la función que invoca a la función objetivo o devuelto desde la función objetivo a la función que invoca. El paso de parámetros permite transferir información entre subprogramas, existen dos tipos de paso de parámetros:

1. Por Valor
2. Por Referencia

Paso de Parámetros por Valor

- La función recibe una copia de los parámetros que se han utilizado para la llamada (parámetros actuales); por lo tanto, los cambios que se produzcan en ellos durante la ejecución del subprograma no afectan el valor de los parámetros actuales.

```
Void  DibujarPuntos(int num){
 int i;
 for (i=1;i<=num;i++)
      printf(“.”);
}
Int main(){
      DibujarPuntos(30);
      return 0;
}
```



Paso de Parámetros por Valor

```
#include<stdio.h>
void DibujarPuntos(int);
int main(){
        int n;
        n=30;
        DibujarPuntos(n);
        printf("%d",n);
        return 0;
}
void DibujarPuntos(int num){
        int i;
        for (i=1;i<=num;i++)
                printf(".");
        num=10;
}
```

Paso de parámetros por valor

```
#include<stdio.h>
Void modificar(int a){
        a=a*3;
        printf("valor desde el
procedimiento: %d",a);
}
Int main( ){
        int dato;
        dato=2;
        printf("antes de la llamada:
%d",dato);
        modificar(dato);
        printf("luego de la llamada:
%d",dato);
        return 0;
}
```

Variable / Lugar	dato	a	Salida
Prog.Principal	2	-	2
Modificar	2	2*3	6
Prog. Principal	2	-	2

Paso de Parámetros por Referencia

Lo que se pasa al subprograma es la dirección de memoria del parámetro actual, de esta forma si la variable pasada es cambiada dentro del subprograma también lo hace dentro del programa principal.

```
#include<stdio.h>
Void modificar (int *);
Int main( ){
      int dato;
      dato=2;
      printf("antes de la llamada: %d",dato);
      modificar(&dato);
      printf("luego de la llamada: %d",dato);
      return 0;
}
Void modificar(int *a){
      *a*=3;
      printf("valor desde el procedimiento: %d",*a);
}
```

Variable Lugar	dato	a	Salida
Prog.Principal	2	-	2
Modificar	2*3	2*3	6
Prog. Principal	6	-	6

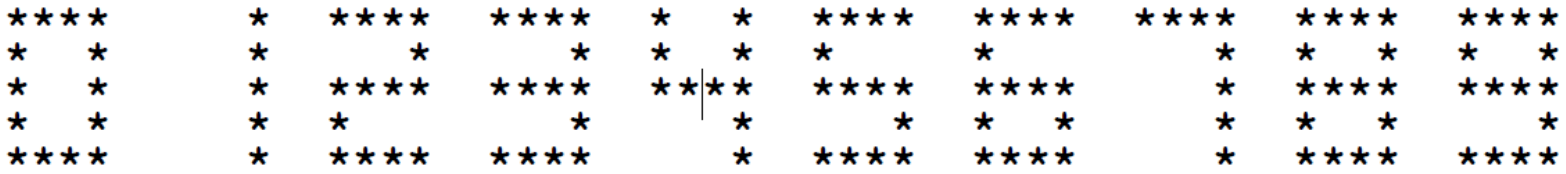
Ejercicios

1. La “prueba del 9” de un número natural consiste en obtener la suma módulo 9 de los digitos del número; es decir, si al sumar un dígito se obtiene un valor mayor o igual que 9, se resta 9. Desarrolle una función que devuelva “la prueba del 9” del número suministrado, sin utilizar la operación mod 9, y un programa que compruebe si prueba_del_9(n) coincide con n mod 9.
2. Diseñe una función que devuelva un valor verdadero si un número de cuatro cifras cumple que: la suma del primer y tercer digito es igual a la suma del segundo y cuarto dígito. Realice un programa que utilice esta función para encontrar todos los números de 4 cifras que cumplen esta condición.
3. Diseñe una función comb(m,n) que calcule el número combinatorio. Escriba un programa que lea números naturales y muestre su combinatorio.

$$\binom{m}{n} = \frac{m!}{n!\,(m-n)!}$$

Ejercicios

1. Crear un procedimiento que intercambie los valores de dos variables. Realizar un programa que llame este procedimiento para que intercambie dos valores leidos desde teclado y los muestre por pantalla.
2. Desarrollar los procedimientos mostrar_0,mostrar_1,mostrar_2,…,mostrar_9 que visualicen por pantalla los números 0,1,2,…,9 respectivamente en tamaño grande, tal y como se muestra a continuación:



1. Para comprobar los procedimientos desarrollados, escriba un programa que obtenga iterativamente un número natural y muestre por pantalla sus dígitos (uno debajo del otro) con el formato anterior.
2. Dado un número real **x** y un entero positivo **d**, se desea obtener el resultado de evaluar un polinomio en **x** como el siguiente:

$$d_n\, x^1 + d_{n-1}\, x^2 + d_{n-2}\, x^3 + \ldots + d_1\, x^k$$ donde cada $\mathbf{d_i}$ es un dígito de **d**

Ejemplo: si d = 6204 el polinomio será: $4\, x^1 + 0\, x^2 + 2\, x^3 + 6\, x^4$

a) Indique que procedimientos son necesarios para resolver este problema, describiendo el uso de los mismos, así como también un detalle de los parámetros que utilizan.

b) Escriba un programa en Pascal para resolver el problema, teniendo en cuenta lo desarrollado en el inciso anterior