MATERIA: **Fundamentos de computadores Digitales**

Funciones, Ambito y alcance, Múltiples fuentes

UNLZ Universidad Nacional de Lomas de Zamora

Facultad de INGENIERÍA UNLZ

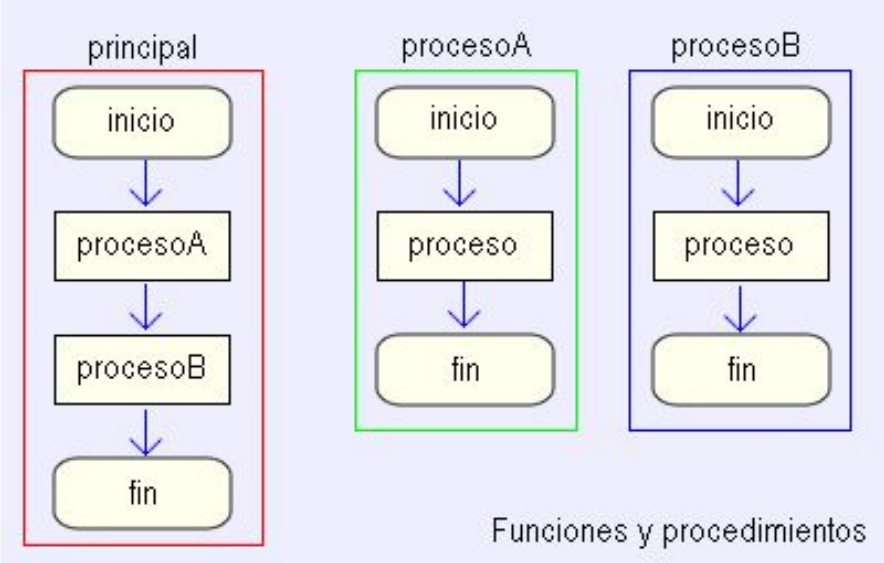
v. 1

Procedimientos y Funciones

Los programas se pueden subdividir en “partes” genéricamente llamadas rutinas, la que suelen clasificarse como:

- Procedimientos: no devuelven valores
- Funciones: calculan y devuelven un resultado

Su principal ventaja es la reutilización de código y facilidad de lectura



Procedimientos y Funciones

En programación, una función es una sección de un programa que calcula un valor de manera independiente al resto del programa.

En esencia, una función es un mini programa: tiene una entrada, un proceso y una salida.

Una función tiene tres componentes importantes:

-Los parámetros, que son los valores que recibe la función como entrada;

-El código de la función, que son las operaciones que realiza la función; y

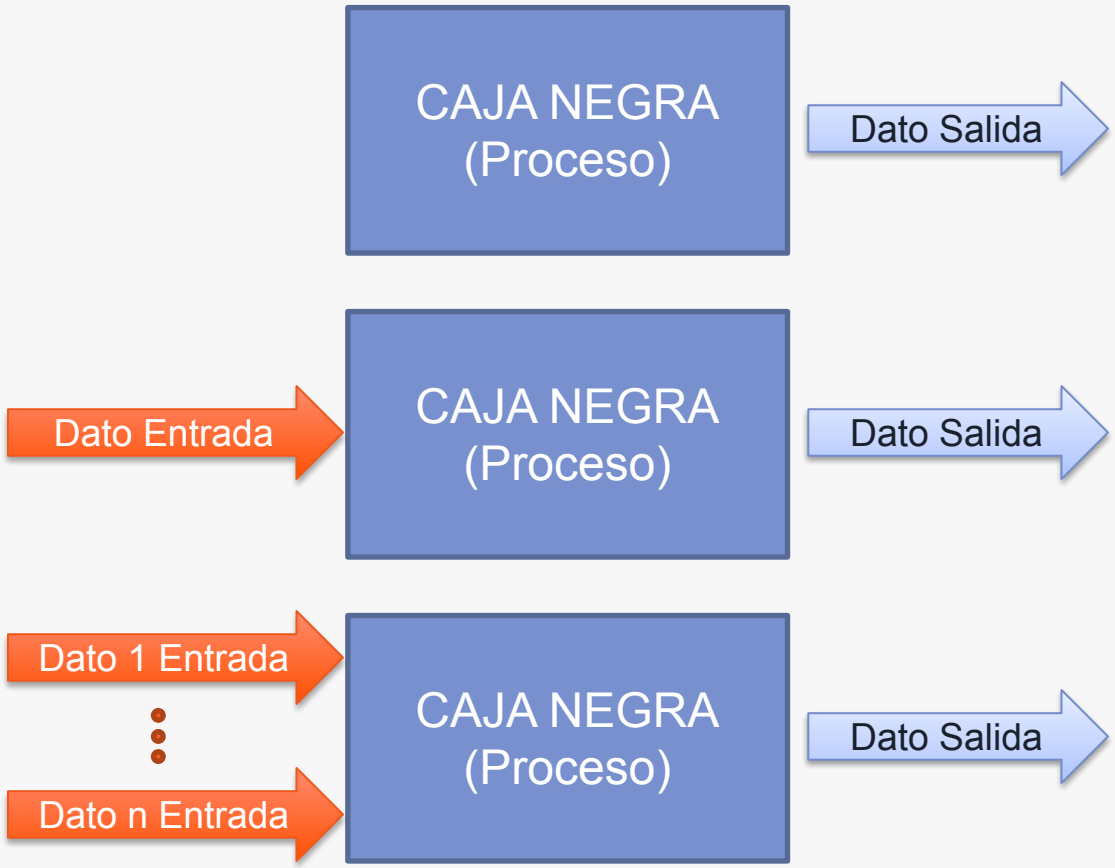
-El resultado o valor de retorno, que es el valor final que entrega la función (return)

Un procedimiento es una sección de un programa (al igual que una función) que realiza varias sentencias de manera independiente al resto del programa. La diferencia con una función es que un procedimiento no entrega ningún valor como resultado, pero **sí** puede recibir parámetros.

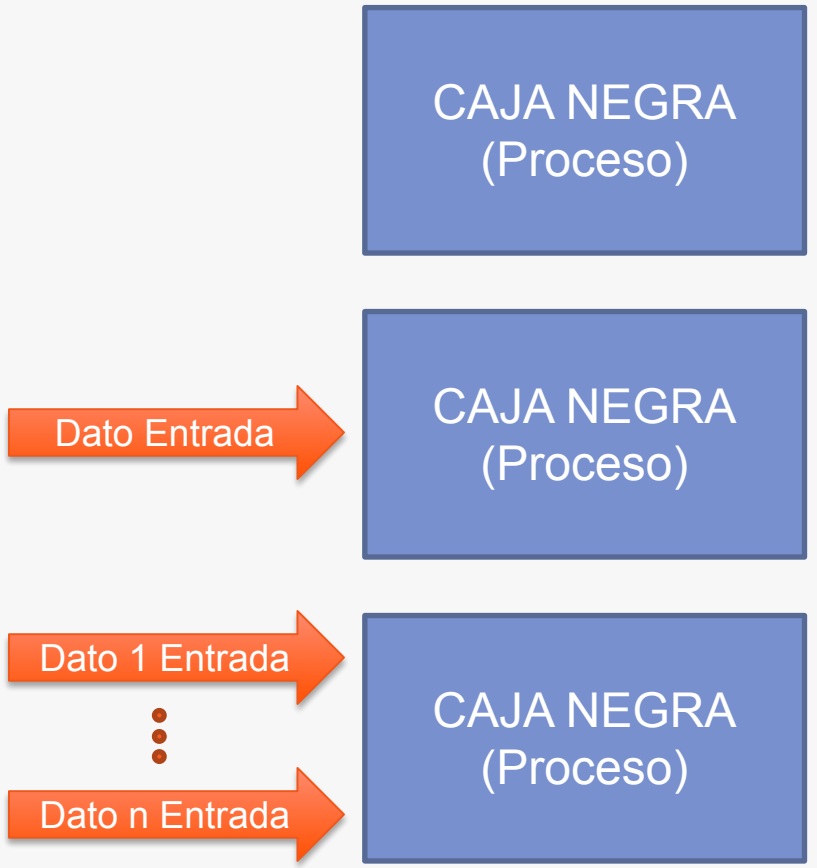
Aunque hay diferencias, se suele llamar a ambas como **funciones** en los libros y apuntes.

Procedimientos y Funciones

Funciones



Procedimientos



Declaración e implementación

Los procedimientos y funciones se **DECLARAN** e **IMPLEMENTAN.**

La **declaración** sirve para determinar el nombre del procedimiento o función, los parámetros que recibe, y el tipo de dato que devuelve:

- VOID si es un procedimiento
- Cualquier tipo de dato si es una función (int, float, double, char, bool, etc)

La **implementación** sirve para definir qué trabajo realiza y cómo lo lleva a cabo.

Se puede declarar y luego implementar, o hacer ambas a la vez.
La sintaxis general de una declaración es:

tipo-que-devuelve nombre (**tipo-parámetro-1** nombre-parámetro-1, …, **tipo-parámetro-n** nombre-parámetro-n);

EJEMPLO DECLARACIÓN: **int** suma (**int** numeroA, **int** numeroB);

Declaración e implementación

En la **implementación**, como dijimos anteriormente, debe ir todo código correspondiente al proceso o tarea de dicha función. Dicho código debe ir dentro de un juego de llaves { }.

Cuando es un procedimiento, es decir, no hay dato de salida, un ejemplo sería:

```
void  mostrar_numero  (int  numero){
   printf("El numero es: %d", numero);
}
```

Cuando la función debe encargarse de retornar un valor de salida, debe usarse la palabra reservada **return.** El dato de salida debe coincidir con la declaración previa, y solo puede devolver UN DATO:

```
float multiplicar_por_dos  (float  numeroA ){
   float resultado = numeroA * 2 ;
   return resultado;
}
```

Ejemplo

Archivo main.c :

```
#include <stdio.h>

double division(double dividendo, double divisor){
    return dividendo/divisor;
}

int main()
{
    double a = 21.32;
    double b = 4.1;
    double z;

    z = division(a, b);
    printf("%4.2f / %4.2f = %4.2f\n", a, b, z);
    return 0;
}
```

Declaración e implementación

Ejemplo

Archivo main.c :

```
#include <stdio.h>

double division(double dividendo, double divisor);

int main()
{
    double a = 21.32;
    double b = 4.1;
    double z;

    z = division(a, b);
    printf("%4.2f / %4.2f = %4.2f\n", a, b, z);
    return 0;
}

double division(double dividendo, double divisor){
    return dividendo/divisor;
}
```

Declaración

Implementación

¿Cómo invocamos una función?

Lenguaje C reconoce que se está haciendo uso de una función o procedimiento cuando junto al nombre de ella se pone un juego de paréntesis (), donde de ser necesario irían los argumentos de entrada:

```
nombre_de_funcion ();
nombre_de_funcion (nombre_variable_1);
nombre_de_funcion (nombre_variable_1, nombre_variable_2);
```

Cuando no hay dato de salida

```
resultado = nombre_de_funcion ();
resultado = nombre_de_funcion (nombre_variable_1);
resultado = nombre_de_funcion (nombre_var_1, nombre_var_2);
```

Cuando hay un dato de salida. **Deben coincidir los tipos de datos**

¿Cómo les pasamos los parámetros?

Los parámetros o **argumentos de entrada** son los datos que necesita la función/procedimiento para hacer su trabajo, y vienen de “afuera”, es decir, son datos que se encuentran en el código principal, en el cual nosotros hacemos uso de la función/procedimiento.

El mecanismo para pasar datos entre el código que “llama” a la función y el invocado puede clasificarse de la siguiente manera:

- **Paso por valor**: Se crea una copia local de la variable dentro de la función.
- **Paso por referencia**: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Ejemplo

Archivo main.c :

```
#include <stdio.h>

double division(double dividendo, double divisor){
    return dividendo/divisor;
}

int main()
{
    double a = 21.32;
    double b = 4.1;
    double z;

    z = division(a, b);
    printf("%4.2f / %4.2f = %4.2f\n", a, b, z);
    return 0;
}
```

Declaración e implementación

Invocación a la función

Salida:

```
21.32 / 4.10 = 5.20
```

Procedimientos y Funciones

En lenguaje C **todo** es una función, incluso nuestro programa principal es una función llamada “main”:

```
int main {
   return 0;
}
```

También, y sin saberlo, estuvimos haciendo uso de funciones preestablecidas como por ejemplo: printf() y scanf()

Si cambiamos el orden...

Archivo main.c :

```
#include <stdio.h>

int main()
{
    double a = 21.32;
    double b = 4.1;
    double z;

    z = division(a, b);
    printf("%4.2f / %4.2f = %4.2f\n", a, b, z);
    return 0;
}

double division(double dividendo, double divisor){
    return dividendo/divisor;
}
```

Salida:

warning: implicit declaration of function 'division' (Declara implícitamente la función y presupone que devuelve int)

error: conflicting types for 'division' here (devuelve double en vez de entero)

note: previous implicit declaration of 'division' was

Explicación

- El problema en el ejemplo anterior es que invocamos una función (división) que no fue declarada previamente.
- Todo identificador es reconocido desde el punto de su declaración (o definición) hasta el final del bloque donde fue declarado
- Como en lenguaje C no se permiten declarar funciones anidadas (declarar una función dentro de otra) todas las funciones están declaradas a nivel de archivo
- Las variables se pueden declarar a nivel de archivo, es decir afuera de toda función, se las conoce como variables estáticas (“globales”). O dentro de una función, conocidas como variables automáticas (locales)
- Incluso, una variable puede declararse dentro de un bloque interno a una función.

Solución

Archivo main.c :

```
#include <stdio.h>

double division(double dividendo, double divisor);

int main()
{
    double a = 21.32;
    double b = 4.1;
    double z;

    z = division(a, b);
    printf("%4.2f / %4.2f = %4.2f\n", a, b, z);
    return 0;
}

double division(double dividendo, double divisor){
    return dividendo/divisor;
}
```

Declaración

Implementación

Salida:

21.32 / 4.10 = 5.20

Ámbito y Alcance

- El **ámbito** (scope) de un identificador es la porción de código en la cuál está asociado a un determinado objeto (variable, función, etc). Esto define su **alcance**, es decir, en que partes del programa el identificador puede ser usado
- Las variables declaradas dentro de un bloque son por defecto automáticas. Son creadas cuando el programa ingresa en el bloque que las define y se destruyen al salir del mismo. Al crear la variable su valor es indeterminado, el programador debe asignar un valor explícitamente
- Las variables declaradas a nivel de archivo (fuera de toda función) son estáticas y existen durante toda la ejecución del programa. Si el programador no las inicializa explícitamente son puestas “a cero” antes de comenzar el mismo

Facultad de INGENIERÍA UNLZ

```c
#include <stdio.h>

int funcion(int x);

int global = 5;

int main()
{
        int local_x = 2;
        int local = 1;

        printf("En main: local = %d\tlocal_x = %d\n", local, local_x);
        local_x = funcion(local_x);
        printf("En main: local = %d\tlocal_x = %d\n", local, local_x);

        while(--local_x) {
                int local = 2;
                printf("En while: local = %d\tlocal_x = %d\n", local, local_x);
        }
        printf("En main: local = %d\tlocal_x = %d\n", local, local_x);
        return 0;
}

int funcion(int x)
{
        int local = 2;
        printf("En funcion: local = %d\tx = %d\n", local, x);
        return local * x;
}
```

Salida

Múltiples Fuentes

Dado que es común usar múltiples fuentes, ¿Cómo hacemos entonces para usar funciones que están implementadas en otro archivo fuente ?

Igual que como hacemos para usar, por ejemplo, printf, usando la directiva del preprocesador #include

#include referencia un archivo encabezado, que se estila guardar con la extensión .h

El resultado es como si la línea que contiene la directiva #include fuera reemplazada por el contenido del archivo que menciona.

Lo habitual es que cada fuente archivo.c tenga su correspondiente encabezado archivo.h que contiene las declaraciones de las funciones implementadas en su fuente, de modo que se lo pueda incluir en otros fuentes:

- ✔ Archivos fuente (.c): definición/implementación de funciones y variables
- ✔ Archivos de encabezado (.h): Declaraciones de funciones y variables

Ejemplo

Archivo main.c :

```
#include <stdio.h>
#include “mate.h”

int main()
{
        double a = 21.32;
        double b = 4.1;
        double z;

        z = division(a, b);
        printf("%4.2f / %4.2f =” “%4.2f\n",
                a, b, z);
        return 0;
}
```

Archivo mate.h :

```
#ifndef MATE_H_INCLUDED
#define MATE_H_INCLUDED

double division(double dividendo, double divisor );

#endif // MATE_H_INCLUDED
```

Archivo mate.c :

```
double division(double dividendo, double divisor ) {
        double resultado = dividendo/divisor;
        return resultado ;
}
```