

## Programación I

Teoría: María Eugenia Marquetti;

martes y viernes de 3 a 5 de la tarde

marielanabel85@herrera.unt.edu.ar.

### Objetivos generales:

- Conocer las metodologías de programación
- Desarrollar procesos de solución de problemas de diversa índole y de complejidad creciente.
- Desarrollar competencias intelectuales y prácticas.

### Objetivos específicos

- Desarrollar sus capacidades creativas en el marco de la metodología enseñada para resolver el problema
- Socializar las estrategias de soluciones de los problemas
- Usar un lenguaje de programación de alto nivel que permita acceder de forma directa a los lenguajes y meta lenguajes emergentes
- Codificar en el lenguaje elegido de los algoritmos
- Usar de manera eficiente los recursos de programación

### Bibliografía

- Gottfried B. Programación en C 
- Kernighan B.W. y P.M. Ritchie El lenguaje de programación C.
- Schildt H. Manual de lenguaje C
- Joyanes Aguilera L. y Zahonero Martínez I. Programación en C

No es promocional  $\wedge$   $\neg$   
para regularizar en ambos parciales nota  $\geq 5$

Hay que rendir examen final

### Unidades temáticas

- Unidad 1: Modelización y especificación de problemas
- Unidad 2: Programación estructurada
- Unidad 3: Tipos de datos
- Unidad 4: Estructuras de selección
- Unidad 5: Estructuras iterativas.
- Unidad 6: Funciones
- Unidad 7: Funciones definidas por el usuario
- Unidad 8: Tipos de datos derivados e indexados
- Unidad 9: Tipo de dato Puntero.
- Unidad 10: Tipos de datos derivados estructuras
- Unidad 11: Archivos en Disco

### Estructuras algorítmicas fundamentales.

1. Secuencias
2. Selección
3. Iteración

Asignación  $a \leftarrow 15$

Ler. (segmento)

ESCRIBIR (argumento)

- OP. aritméticas + - \* / resto.
- Seleccionar  $\leftarrow$
- repetir  $\leftarrow$

## Selección

Si cond ENTONCES

A

(FIN Si) una rama

Si cond ENTONCES

A

SINO

B

(Fin Si) dos ramas.

SEGUN valor .

cond : A1

cond : A2

:

cond : An

SINO

An+1

(Fin SEG) estructura según

## Iteración

MIENTRAS cond

A

(Fin MIENTRA)

HACER n VECES (i = vi, vf, ...)

A

(Fin HACER)

REPETIR

A

HASTA cond

Ejercicio 2: Considere una frase como una se cuenda de caracteres con marca final. Las palabras de la frase están separadas por uno o mas blancos. Escriba un algoritmo que le permita procesar la frase y calcular la cantidad de palabras que terminan en "s" o "r". MF: " "

ALGORITMO: FRASE

ENTRADA: CC: carácter

SALIDA: cantpals, cantpalr: enteros

A<sub>1</sub>: INICIALIZAR

A<sub>11</sub>: cantpals ← 0

A<sub>12</sub>: cantpalr ← 0

A<sub>13</sub>: LEER(CC).

A<sub>2</sub> MIENTRAS (CC ≠ ".")

Si (CC = "s") ∨ (CC = "r")

C1 ← CC

• LEER(CC)

C2 ← CC

Si (C1 = "s")

Si (C2 = ".") ∨ (C2 = "s")

    cantpals ← cantpals + 1

Si (C1 = "r")

    cantpalr ← cantpalr + 1

FIN MIENTRAS.

A<sub>3</sub>: ESCRIBIR(cantpalr, cantpals)

A<sub>4</sub>: PARAR.

## Unidad I

- El proceso de programación: Etapas en la resolución de problemas con computadora.
- Algoritmos: formas de reducción de complejidad de problemas del mundo real.

### Preguntas iniciales

- Como se resuelve un problema del mundo real con computadora?
- Como se expresa la solución?
- como se reduce la complejidad del problema.

El programador debe realizar algunos procesos intelectuales.

1 Abstracción

2 Modelización del problema

3 Especificación del problema real

4 Escritura de la solución como una expresión ejecutable en la computadora

1. Analizar el mundo real, e interpretar los aspectos esenciales de un problema y expresarlo en términos precisos

2. Encontrar los aspectos principales y los datos que se habrán de procesar y el contexto. Simplificar su expresión

3. El proceso de analizar los problemas del mundo real y explicar) determinar en forma clara y concreta el objetivo que se desea.

4- Es la escritura de un programa que represente una solución ejecutable en una computadora, usando un lenguaje de programación

El algoritmo debe ser la solución al problema planteado.  
 El algoritmo es la especificación rigurosa de la secuencia de pasos (instrucciones) para resolver un problema.

La biblio: Algoritmos + estructuras de datos = programas.  
 Escritor de Wirth.

Niklaus Wirth, científico de la computación.

### Verificación y documentación

- Pruebas (testing)
- Depuración
- Alternativas de diseño y estilo.

① Dada la siguiente sucesión de números enteros:

1 3 6 10 15 21 28 36 45

Diseñe y escriba un algoritmo modularizado que,

- determine la regla de formación de cada término
- genere la sucesión de  $N$  términos

$$\begin{array}{cccccccccc} 1 & \checkmark & 3 & \checkmark & 6 & \checkmark & 10 & \checkmark & 15 & \checkmark \\ 1 & 2 & 2 & 3 & 3 & 4 & 4 & 5 & 5 & 6 \\ & & & & & & & & & 6 \\ & & & & & & & & & 7 \\ & & & & & & & & & 7 \\ & & & & & & & & & 8 \\ & & & & & & & & & 8 \\ & & & & & & & & & 9 \end{array}$$

ALGORITMO: serie  $\rightarrow$  aux.  $\rightarrow N$

ENTRADA:  $N$ , entero  $> 0$

SALIDA: num, entero  $> 0$  (num de la serie).

VARIABLES AUXILIARES: i, entero  $> 0$ , aux, entero

- INICIALIZAR  $\rightarrow$  num  $\leftarrow 1$
- LEER ( $N$ )  $\rightarrow$  aux  $\leftarrow 0$

- generar los números de la serie y escribirlos - ②

- PARAR

② HACER  $N$  VECES ( $i = 1 \dots N$ )

$$\rightarrow aux \leftarrow i + 1$$

$$num \leftarrow num + aux$$

NOTA ESCRIBIR (num)  
 (fin de HACER)

## Unidad 2

- Lenguajes de programación
- Programación estructurada.
- El lenguaje C
- Elementos básicos de un programa.
- Análisis de programas
- Estilo de programación

### Lenguajes de programación

- Los 5 lenguajes de programación más populares en la actualidad son:
- Java, C, C++, Python, C# según la página TIOBE.

tipado: algo que tiene tipo de dato. (adjetivo).

Java

Reconocido por su legibilidad y simplicidad

Es uno de los lenguajes de programación más adoptados.

Su enorme popularidad se debe a su poder de permanencia, cosa que asegura el funcionamiento a largo plazo de las aplicaciones que lo utilizan.

C

Es uno de los más utilizados en el mundo

Sí bien es ejecutado en la mayoría de los sistemas operativos, es de propósito general, con lo cual es muy flexible.

Muy utilizado para el desarrollo de aplicaciones de escritorio, como el popular editor gráfico GIMP

## C++

Conocido como "C plus plus", este lenguaje de programación orientado a objetos surge como una continuación y ampliación del C. Hay una gran cantidad de programas escritos en C++ como por ejemplo los paquetes de Adobe.

## C#

"C sharp", lenguaje de programación orientado a objetos, desarrollado en el año 2000 por Microsoft para ser empleado en una amplia gama de aplicaciones empresariales ejecutadas en el framework .NET.

C sharp es una evolución del C y C++ que se destaca por su sencillez y modernidad.

## Python

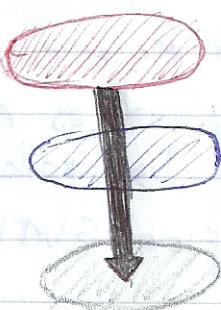
Lenguaje de programación interpretado. Hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma.

Soporta orientación a objetos, programación imperativa y programación funcional.

## La programación estructurada.

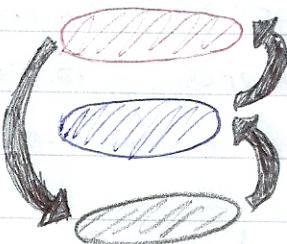
Lo clásico de la programación estructurada se refiere al control de ejecución.

Es una de las cuestiones más importantes que hay que tener en cuenta al construir un programa en un lenguaje de alto nivel.



Los lenguajes de programación más antiguos se apoyaban en una instrucción de transferencia incondicional de su control (con la instrucción "go to" que significa "ir a").

Consecuencia: programas poco legibles y difíciles de comprender.



### La segmentación

Es la visión moderna de la programación estructurada.

Segmentos o módulos que se habrán de integrar.

Una regla práctica para lograr este propósito es establecer que cada segmento del programa no excede, en longitud, de una página de codificación, o sea, alrededor de 50 líneas.

### De lo abstracto hacia lo práctico:

Desarrollar programas usando la técnica llamada refinamiento sucesivo.

El problema inicial es segmentado.

Principio de Divide y conquistarás.

La integración debe ser sencilla y no presentar problemas en sí misma.

La programación estructurada propone separar los procesos en estructuras elementales

- \* Secuencia
- \* Selección
- \* Iteración.

## Teorema del programa estructurado.

• Los programas que utilizan sólo estas tres instrucciones de control básicas o sus variantes se llaman estructurados y tienden a lo práctico.

• ¿Y como es un programa NO estructurado?

• Es un programa con transferencias incondicionales que dificulta el trabajo de los programadores y la integración del trabajo en común.

Toda función computable puede ser implementada en un lenguaje de programación que combine solo las tres estructuras lógicas (también llamadas estructuras de control).

Este teorema demuestra que la instrucción GOTO, no es estrictamente necesaria y que para todo programa que la utilice existe otro equivalente que no hace uso de dicha instrucción.

Se le atribuye el teorema a Corrado Böhm y Giuseppe Jacopini

• Así la visión moderna de un programa estructurado es un compuesto de segmentos, los cuales pueden estar constituidos por una pocas instrucciones o por una página o más de código.

Cada parte englobará funciones y datos relacionados semánticos o fundamentalmente.

## Lenguaje de Programación C

Creado en la década de los 70

Los autores del lenguaje son Brian Kernighan y Dennis Ritchie.

En un programa intervienen:

Ordenes para el preprocesador

VARIABLES

CONSTANTES

ARITMÉTICA

CONTROL DE LA EJECUCIÓN

FUNCIÓNES

PROCEDIMIENTOS DE ENTRADA Y SALIDA

COMENTARIOS

PSEUDO

ESCRIBIR (" ") = printf ("");

LEER (" ") = scanf ("%d", &dato);

DECLARACIÓN VARIABLES = int dato, resultado;

DEFINIR CONSTANTES = #define NUM 189

V ← 3 = V = 3; ;

PARAR = return 0;

El bloque de definición del programa se cierra entre {}

{  
    código  
};

#include <libreria.h> se utiliza para ingresar librerías.

<stdlib.h> librería estandar.

<stdio.h> librería estandar input, output. Habilitan las funciones de escribir y leer.

orden → nombre → valor  
 → #define CONSTANTE 150 // se define una constante.

(Las constantes con letras mayúsculas)  
 tipo → función → argumento (vacío) → void = vacío.

→ int main (void). ← una función que aparecerá en el principio de TODOS LOS CÓDIGOS EN C. es lo más importante. Marca el inicio de bloque de definición.

Hay palabras reservadas por el lenguaje.

Otras funciones:

float sqrt (x) → retorna flotante

int sqrt (x) → retorna valor entero

Para declarar variables:

int dato, resul;  
 ↓  
 tipo (entero) nombre  
 variables.

Cada instrucción se delimita con ;

Código de ejemplo

```
#include <stdio.h> // usar funciones Entrada Salida
#define CONSTANTE 150 // define la constante
```

```
int main(void) // función principal
{ int dato, resul; // declarativa de variables
```

```
setvbuf (stdout, NULL, _IONBF, 0); // I/O de I/O
```

```
printf ("ingrese el numero dato. GRACIAS \n"); // función escribir
```

```
scanf ("%d", &dato); // función LEER
```

```
resul=dato * CONSTANTE;
```

```
printf ("El producto es %d", resul);
```

```
return 0; // PARAR el código en la memoria
```

}

### De las variables y las constantes

- Objetos sobre los que actúan las instrucciones que componen el programa.
- Deben declararse como tales.  
Deben tener un identificador asociado, formado por una secuencia de letras y números. El 1º carácter debe ser una letra. Letras mayúsculas y minúsculas son diferentes.
- Debe indicarse que tipo de datos pueden contener (rango y operación permitidas)
  - Las variables pueden inicializarse en forma grupal
  - La declaración de las variables reúne todas estas exigencias
  - Las constantes (constantes simbólicas) permiten hacer modificaciones sistemáticas.
  - Las constantes se definen en una linea `#define`.  
`#define NOMBRE valor.`
  - Las constantes pueden ser enteras, reales y de carácter.
  - Deben definirse previo a su uso.

### Aritméticas:

Tipos de datos básicos: `int`, `float`, `char`, `short`, `long`, `double`.

→ permite alargar el tipo de datos

Interacción entre los operadores aritméticos y las variables y/o constantes declaradas. (si el tipo es numérico).

El tipo de operación permitida está ligado al tipo de dato con que fue declarada la variable y/o constante

## Iteración

## Proposición for

for (variable de control = valor inicial; condición; variación de la variable de control)

proposición;

$i + i$  es lo mismo.

```
for (i=1; i<=12; i=i+1) //ejemplo
{
    res=num+i;
    printf ("%5d", res)
}
```

Notas: el printf se puede reemplazar por puts() cuando lo que se escribe no tiene formato.

Selección:

if - else

if (condición)

proposición 1;

else

proposición 2;

# pdfelement

Proposición puede ser otra proposición if-else

Iteración:

while (condición)

proposición;

do while

do {

proposición;

} while (condición)

Comentarios // una linea /\* varias lineas \*/

(Un programa en una sola linea)

#include <stdio.h>

int main (void)

{printf("Hola. PODE2016 de una sola linea\n"); /\*programa\*/ return 0; }

Tipos y Tamaños de datos.

Tipos de datos básicos.

float

4 bytes

char

1 byte

double

8 bytes

int

2 bytes

Aritmética módulo 2:

int

2 o 8 bytes

$2^n$ , n; numero de bits.

8 bits = 1 byte

n = 2 bytes.

$2^{16} \rightarrow 65535$

$2^{16}-1 \rightarrow -32768, 32767$

Calificadores para los tipos enteros. (short) y (long)

3 tipos enteros: short int → 16 bits (2 bytes)  
 int → 16 o 32 bits.  
 long int → 32 bits (8 bytes)

short < int < long

se puede omitir el int y dejar solo short y long.

Otros calificadores

Signed

Unsigned

Los números unsigned obedecen a la aritmética modulo  $2^n$  con n numero de bits.

Si los char son de 8 bits, las variables unsigned char toman valores de 0 a 255. (\*)

Iteración - Proposición for

inicialización      bucle      incremento

for (expresión1; expresión2; expresión3);

Proposición

Los componentes del for son 3 expresiones.

Generalmente expresión1 y expresión3 son asignaciones, y expresión2 es una condición

Se puede omitir cualquiera de ellas.

Ver notas del break asociado con for y while. K&R pag 68

expresión1

for (; expresión2; expresión3)

proposición

expresión1

for (; expresión2;)      for (;;)

proposición

expresión3

proposición

↑  
probar

Ejemplo

int i = 1;

for (; i < 15, i++)

Ejemplo de salida de un programa que usa for anidados

123456321

1234 6321

123 321

12 21

1 1

Ejemplos: (\*) a

b.

int 16. -32768, 32767. unsigned int 16 0 a 65535, signed int 16 a, short int 16 a unsigned short int 16 b, ... etc (ver tabla)

## Instalar eclipse Helios (3.6)

Iteración - proposición while.

while (expresión)

proposición

La proposición debe tener una inicialización externa y una variación de la expresión para evitar ciclos infinitos.

Si es una sola linea de iteración no es necesario agregar las llaves.

Tratamiento de una secuencia con marca final

Escribir un algoritmo que lee una secuencia de números enteros positivos que termina con un número negativo.

El algoritmo debe contar y mostrar cuantos pares hay en la secuencia.

```
int num, cont;
cont = 0;
scanf("%d", &num);
while (num >= 0)
{
    printf(num);
    res ← num % 2;
    if (res = 0)
        cont = cont + 1;
    scanf("%d", &num);
}
printf(cont)
```

ALGORITMO: Serie de azúcar  
 ENTRADA: num, entero > 0  
 SALIDA: cont, entero > 0  
 VAUX:

## Unidad III

### Tipos de datos

Los elementos que se habrán de usar en el programa, constantes y variables, tienen un tipo de dato asociado en su declaración.

El tipo de dato indica el rango de valores que pueden asumir dichos objetos y las operaciones permitidas para con ellos.

2 grupos de tipos de datos: datos básicos y datos estructurados o compuestos



Compiladores que compatibilizan con el estandar ANSI C

- GCC
- Microsoft Visual C++
- ARM

GCC, compilador integrado del proyecto GNU para C, C++, Objective C y Fortran, la sigla GCC significa "GNU Compiler Collection"

ARM es

## Constantes y tipo asociado

Al definir constantes de tipo numérico se puede indicar el calificador asociado.

Un entero long se escribe con una l o L terminal.

Las constantes sin signo con u o U

También se pueden definir constantes de carácter y de cadena.

constante de cadena

printf ("Ingrese un entero \n")

único caso en el que una parte del constante de cadena es un constante de carácter

Secuencias de escape.

Tipo de dato. float

float ← 32 bits

double ← 64 bits

long double ← permite trabajar con 12 dígitos después del punto.  
(64 o 80 bits según versión).

Operador ternario: ?

$z = (a > b) ? a : b;$

printf ("z=%d.%d", z);

En C

Falso - 0

Verdadero ← 1 (o cualquier valor distinto de 0 en algunos casos de iteración)

Otra organización según

Evitar ambigüedades

if ( $a > b$ )if ( $b > c$ ) $z = a$ 

else

 $z = b$ if ( $a > b$ )if ( $b > c$ ) $z = a$ 

else

 $z \neq b$ 

→ TRAMPA! por falta

de llaves se asocia

al if anterior  
(if ( $b > c$ )).

Otras organizaciones según la lógica

del problema

pdfelement

if ( $a > b$ ) {if ( $b > c$ ) { $z = a$ 

} else

 $z = b$ 

}

}

con las llaves evitamos ambigüedades.

como la anterior

AN

En switch.

Se pueden utilizar dos proposiciones al final de cada case

\* Proposición break:

Provooca una salida inmediata del ciclo iterativo. El ciclo que la contiene termina

\* Proposición continue:

provocea que se inicie la siguiente iteración del ciclo que la contiene

Por ejemplo

```
HACER n VECES // i = 1, n
{
    prop1; ✓
    prop2; ✓
    Si (resto(i, 2)) = NT.
        continue
    (Fin Si)
    prop3;
}
prop4
```

si entra en el si y llega al  
continue, vuelve al hacer con  
un valor de  $i = 2$  y no realiza  
la prop 3. Hace que continue  
el proceso iterativo.

HACER n VECES // i = 1, n

```
{
    Prop1;
    Prop2;
    Si (resto(i, z)) = NT
```

(Fin Si)

Prop3;

}

Prop4;

el break hace que se salga  
del proceso iterativo. No realiza  
la prop3 y pasa a la instrucción  
inmediatamente después del fin  
de hacer, en este caso la prop4

## Tipos de dato carácter

Tipos de datos básicos.



`char = 1 bit = 1 byte.`

UN CARÁCTER

Almacenan solo 1 carácter. (`char letra;`)

SE TRABAJA

Declaración e inicialización: `char letras = 'a';` COMILLAS SIMPLES

SIEMPRE CON

ASCII: American Standard Code for Information Interchange Estándar, código estandar para el intercambio de información.

Creado en 1963 por el comité estadounidense de estándares. Este organismo cambió su nombre en 1969 por Instituto estadounidense " o "ANSI"

- El código ASCII permite la unificación de criterios, así, un archivo de texto escrito en un procesador de palabras de cualquier marca y modelo.

①

Ler y escribir un carácter

②

Con formato %c en una

invocación a la función scanf.

invocación a la función printf

`char letra;`

`scanf ("%c", &letra);`

`char letra;`

`printf ("%c", letra);`

③

`Letra = getch();` ← toma el carácter introducido por el teclado,  
`putchar (letra);` ← lo almacena en la variable letra  
                           ← escribe en el monitor el carácter almacenado en la variable letra

char indice;

for (indice = 122 ; indice >= 97 ; indice --)

putchar (indice) → escribirá la z minúscula, porque  
en ASCII 122 = 'z'

↳ constantes de carácter también se definen con #define.

Constante de un carácter usa "", una cadena de caracteres  
usa ""

## Unidad VI

Un programa escrito en código C es un conjunto de variables y funciones... lideradas por main.

Un programa puede residir en varios archivos fuente, compilados independientemente y cargarse junto con funciones de biblioteca.

La función encapsula cálculos y resguarda la información

### 2 tipos de funciones.

- ① Funciones de biblioteca: printf, scanf, sqrt, isalpha.
- ② Funciones definidas por el usuario: int dividir(int a, int b); char esLetra (char car);

- ①
- No forman parte del lenguaje
  - Disponible en todas las implementaciones
  - Cada compilador de C incluye un repertorio de funciones.
  - Eje: <stdio.h>, <math.h>, <ctype.h>.

② Funciones de uso común de la biblioteca de matemática.

Con  $x$  de tipo double, todas las funciones regresan double.

Los ángulos para las funciones trigonométricas están representados en radianes.

Función	Descripción
sqrt(x)	raíz cuadrada de $x$
exp(x)	función exponencial $e^x$
log(x)	logaritmo natural de $x$ (base e)
log10(x)	logaritmo de $x$ (base 10).
fabs(x)	valor absoluto de $x$

$\sin(x)$  $\cos(x)$  $\tan(x)$ .

⊗ Funciones de uso común en la biblioteca de manejo de caracteres.

int isdigit(int c)

Si (c es un dígito) Entonces regresa V.  
Sino regresa Falso

int isalpha(int c)

Si (c es una letra) Entonces regresa V.  
Sino regresa Falso.

int isalnum(int c)

Si (c es un dígito o una letra) Entonces  
regresa V.  
Sino regresa Falso.

## De módulo a función

- Los módulos tienen propósitos específicos.
- Constituyen una parte distinta y autónoma del programa.
- Se comunican entre sí a través de argumentos y valores regresados por las funciones.
- El argumento es el canal de comunicación entre funciones, en realidad se llamaría "es letra".

ALGORITMO: cuentaLetra.

E: car: carácter

Si  $\rightarrow$  aux: enteroaux  $\leftarrow$  0Si ( $\text{ord}(\text{car}) \leq \text{Z}$ )

aux++

FIN Si

RETORNAR (aux)

int cuentaLetras (char letras)

{

int aux=0;

if ( $\text{ord}(\text{letra}) \geq \text{A}$  &  $\text{ord}(\text{letra}) \leq \text{Z}$ )  
aux++;

return (aux);

}

La proposición return es el mecanismo para que la función regrese un valor a su invocador

FUNCION getchar(). (es lo mismo que el scanf pero mas simple)  
Toma el ultimo caracter leido por la entrada estandar.

Analizar cuantas letras tiene la frase "la tarde."

int main (void).

```
{ char car; int cont=0;
car = getchar(); // 1º lectura.
while (car != '.') {
    { cuentaLetras(car);
        if (cuentaLetras (car) == 1)
            cont = cont + 1;
        car = getchar();
    }
}
```

{ } ] Opción 1

cont = cont + cuentaLetras (car); ] Opción 2.

car = getchar(); ]

Sintaxis para definir una función.

Tipo nombre\_func (lista de parámetros) // encabezado.  
{ declaraciones; // sector de declarativas y las locales  
proposiciones; // cuerpo de la función.  
}

El encabezado de la función tiene:

Tipo asociado a la función. Indica el tipo de retorno de la función.

nombre\_func debe ser un identificador válido.

(listo de parámetros) listado de los parámetros formales de la función.

tipo1 p<sub>1</sub>, tipo2 p<sub>2</sub> ..., tipoK p<sub>K</sub>.

El sector de declaraciones de la función tiene:

## Del tipo de retorno.

- La proposición return permite que la función "regrese" un valor al medio que la invoca.

Forma de uso: return expresión;  
los () son opcionales

### A tener en cuenta

- Usar nombres activos para las funciones
- El tipo de dato asociado a la función indica si la función habrá de devolver valores o no.
- Tipo void, no retorna valores. Una función de este tipo "produce efectos".
- Si se omite el tipo de los parámetros, se asume entero.
- La lista de parámetros puede estar vacía.
- Se pueden omitir algunas partes en la función; por ejemplo las declarativas y las proposiciones del cuerpo:

void nada();

int nada() { return; } //son sintácticamente correctos

no regresa ni hace nada, puede servir para reservar lugar al desarrollar un programa

- C NO permite definir una función dentro de otra



Las funciones se definen fuera del main  
Pero se pueden llamar en cualquier parte del código.

Ejemplo

```
1º void tabla (void);  
int main (void)  
{   tabla ();  
    return 0;  
}  
  
2º void tabla (void)  
{   double result, x = 0.5; int i;  
  
3º   for (i=1; i<=20; i++)  
    { result = .sin (x);  
     x = x + 0.5;  
     printf ("El seno () de %lf es %lf\n", x, result);  
    }  
    return;
```

→ no tiene argumento porque es VOID. Y no tiene ningún valor que devolverle al main.

- 1º - se declara la función
- 2º - El main y código
- 3º - se define la función

La void únicamente produce un efecto, no devuelve un valor.

## En resumen, las funciones

- Se definen
- Se declaran
- Se asocian a un tipo de datos
- Se invocan
- Se ejecutan.
- Se pueden usar como factores de una expresión
- Pueden devolver valor/valores.
- Pueden ejecutar tareas sin retornar valores.
- Pueden llamarse o invocarse desde distintas partes de un programa

Las funciones void producen un EFECTO, no retornan un valor.

¿Dónde deben estar las declaraciones de las funciones definidas por el usuario?

Al inicio del programa, antes de main. Las variables globales se definen aquí.

¿Dónde deben estar las definiciones de las funciones?

En cualquier orden, en esta etapa inicial se recomienda que estén luego del cierre de main.

Un tipo particular son los del tipo void (vacío...).

La función no devuelve ningún valor, o cambio, decimos que "produce un efecto"

void dibujar\_figura (int n);

void imprimir\_nombre (char nombre [30])

## Alcance de un identificador

El alcance de un nombre es la parte del programa dentro de la cual se puede usar ese nombre.

Variables locales es lo mismo que variables privadas.

Variables globales es lo mismo que variables públicas.

### Variables locales

Declaradas en el contexto de una función, comienzan a "existir" cuando se invoca a la función y desaparecen cuando la función termina de ejecutarse.

Solo pueden ser usadas por la función que "LAS CONTIENE".

Se reconocen como "automáticas". No retienen sus valores entre dos invocaciones sucesivas a la función, inicializándose.

### Variables globales.

Son externas a todas las funciones.

Pueden ser accedidas por cualquier función y usadas como parámetros a otras para comunicar datos entre funciones. "Existen" en forma permanente.

Conservan sus valores entre distintas invocaciones.

①

## Programación

### Unidad VIII - Arreglos

#### Tipos de datos derivados:

Existe una cantidad conceptualmente infinita de tipos de datos derivados, formados a partir de tipos de datos simples.

Los tipos de datos simples...

Los tipos de datos derivados...

Las estructuras de datos...

• Una estructura de datos es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre sí de distintas modos.

#### Estructuras de datos Clasificación

Según el tipo de información que contienen.

- Estructuras de datos homogéneas
- " " " heterogéneas.

Según asignación de memoria

- Estructuras de datos estáticas
- " " " dinámicas.

En el código C estandar se predefinen los siguientes tipos de datos derivados

\* Arreglos

\* Funciones

\* Apunadores

\* Estructuras

\* Uniones.

## Arreglos

"Un arreglo es una colección de variables ordenadas e indexadas, todas de idéntico tipo que se refieren usando un nombre común"

"El array (también conocido como arreglo, vector o matriz) permite trabajar con una gran cantidad de datos del mismo tipo bajo un nombre o identificador".

### Características de los Arreglos.

- Es un tipo de dato homogéneo
- Es una estructura indexada.
- Es un tipo de dato estático: los arreglos usan un espacio de almacenamiento contiguo.
- El nombre del arreglo es una referencia a la dirección de la 1<sup>o</sup> componente
- El nombre del arreglo es una referencia a la dirección de la 1<sup>o</sup> componente 

Un arreglo puede ser:

De una dimensión (un índice) o De varias dimensiones (varios índices).

Los componentes del arreglo pueden ser de cualquier tipo: caracteres, enteros, reales, punteros, estructuras, arreglos, etc.

(2)

## Arreglos de una dimensión.

- Un vector o arreglo lineal es un tipo de dato arreglo con un índice, es decir, con una dimensión.
- El acceso a los componentes de un arreglo se puede hacer en forma directa, a través del nombre del arreglo y la notación de subíndice que indica la posición.

A tener en cuenta.

Un arreglo de una dimensión con los primeros 5 primeros números pares positivos tiene la forma:

pares: 

2	4	6	8	10
---	---	---	---	----

El acceso directo, vía nombre y subíndice:

pares[0] = 2

pares[2] = 6

pares[9] = 10.

Como se declara un arreglo de una dimensión  
Forma general: tipo nombre[tamaño];

- tipo: puede ser un tipo aritmético (int, float, double), char, puntero, estructura, unión, otro arreglo
- nombre: cualquier identificador válido
- tamaño: expresión constante entre corchetes que define cuantos elementos guarda el arreglo

\* Un arreglo siempre se declara incluyendo entre los corchetes el máximo número de elementos, salvo que inicialice el arreglo al mismo tiempo.

Uso de constantes enteras:

```
#include <stdio.h>
#define TAMA 20
int main (void) {
    int arreglo[TAMA];
}
```

En C no se puede operar (comparar, sumar, restar, etc) con todo un vector o todo una matriz como única entidad. Hay que tratar sus elementos uno a uno por medio de bucles for o while

Los elementos de un vector se utilizan en las expresiones de C como cualquier otra variable

$$a[5] = 0,8;$$

$$a[9] = 30 * a[5];$$

(3)

## ALGORITMO lee\_arreglo.

ENTRADA:  $n$ : entero  $> 0$ ;arreglo de  $n-1$  números enteros;SALIDA: arreglo con  $n-1$  valores leídos.

CONSTANTES: TAMA = 20

VARIABLES:  $i$ , num; num enteros.Si ( $n$  es menor que TAMA) ENTONCES.HACER  $n$  VECES. //  $i=0, n-1$ .

LEER (num)

arreglo  $\leftarrow$  num.

FIN SI

RETORNAR (arreglo cargado con los valores leídos).

#include &lt;stdio.h&gt;

# define TAMA 20,

int main (void)

{ int arreglo[TAMA]; int ind, n;

printf ("ingrese el orden del arreglo  $\leq 20/n$  "); scanf ("%d/n" &n);if ( $n \leq TAMA$ )

{ for (ind = 0; ind &lt; n; ind++)

scanf ("%d", &amp;arreglo[ind]);

}

return 0;

{.

```
#include <stdio.h>
#define TAMA 20
void leer_arreglo (int n, int arreglo [TAMA]);
void escribir_arreglo (int n, int arreglo [TAMA]);
int main (void) {
    int arre [TAMA]; int n
    printf ("ingrese el orden del arreglo ≤ 20\n"); scanf ("%d\n",&n);
    if (n ≤ TAMA)
    { leer_arreglo (n, arre);
        escribir_arreglo (n, arre); }
    Del tamaño del arreglo
```

- La dimensión del arreglo debe ser establecida a priori
- El tamaño o dimensión del arreglo debe ser una expresión constante
- La declaración de un arreglo → una reserva de memoria... Por lo tanto, el tamaño no puede ser una variable ni una expresión variable.
- Usar constantes definidas con #define para indicar el tamaño (Recomendación).

Superar el tamaño máximo declarado puede generar situaciones irregulares con resultados perjudiciales.

Un ejemplo de irregularidad : si estás declarando el arreglo char letra [3].

¿Qué ocurre si se escribe la asignación letra[3] = 'S' ?

letra: [0] [1] [2] ¿ [3] ?  
 | A | L | A |

(9)

letra: [0] [1] [2] ? [3]!

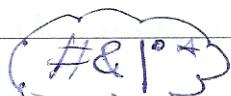
A T L A T I S

Otra situación desgradable:

Si esto declarado el arreglo char letra[3].

¿Qué ocurre si se invoca a la función de escritura con un elemento inexistente?

printf ("%c", letra[35]);



- A costa de la no verificación de los límites del arreglo, los arreglos en C tienen un código ejecutable rápido.

- Se debe ser responsable con la administración del espacio de memoria cuando se trabaja con arreglos.

Ejemplo de un programa que no debe ejecutarse NUNCA:

```
#define LIMITE 10
#define PROBLEMA 500
```

```
int main(void)
{
    int ind, lio[LIMITE];
    for (ind=0; ind < PROBLEMA; ind++)
        lio[ind]=ind;
    return 0;
}
```

## Inicialización de Arreglos

Hay 3 formas de inicializar un arreglo

- Por omisión
- Por inicialización explícita
- En tiempo de ejecución

### (1) Por omisión

Un arreglo declarado en forma global se inicializa por omisión (por default), en ceros binarios, a menos que se indique lo contrario.

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

```
#define MAX50
```

```
int vector [MAX] //ceros binarios, nulo: 0000 0000
```

```
int main (void)
```

```
{ int ind, n, tempor;
```

### (2) Por inicialización explícita

En la declaración, se asignan valores, según la siguiente norma: los valores elementos a ser asignados a los elementos del arreglo deben estar encerrados entre llaves y separados por comas.

```
int vector [5] = { 11, 22, 33, 44, 55};
```

```
float valores - Se mas [3] = { 78.6, 98.9, 45.7, 39.5, 56.7 }
/* situación irregular */
```

(5)

short valores\_de\_menos [5] = { 11, 22, 33 };  
 /\* situación irregular \*/

Si (tipo de comp. es numérico) ENTONCES.  
 elementos restantes  $\leftarrow 0$

SINO

Si (tipo de comp. es carácter) ENTONCES  
 " "  $\leftarrow$  blancos.

• Los corchetes [] pueden estar vacíos?

- Se puede omitir el "tamaño" del arreglo cuando hay una inicialización explícita.
- El tamaño o dimensión se define por la cantidad de elementos dados en la inicialización

### (3) En tiempo de ejecución

• Caso más común

Los componentes del arreglo tomarán valores que son leídos desde algún dispositivo de entrada o sino valores generados por el mismo programa.

### Cadenas - Arreglos de caracteres

- Este es el tipo de arreglo más popular en código C.
- La última celda del vector de caracteres se reserva para el carácter que marca el fin de la cadena, que es el carácter nulo;

"\0"

- El carácter nulo tiene valor ASCII 0000 0000

Inicialización explícita de una cadena

Los valores a asignar deben estar encerrados entre llaves y separados por comas (listo).

#define MAX 45.

int main (void)

```
{ char apellido[MAX] = { "P", "e", "r", "e", "z", "\0" };
```

NO OLVIDAR NUNCA EL TERMINADOR

```
char cadena [] = { 'P', 'o', 'g', '\0', 'z', '0', 't', 'z', '\0' };
```

También vale una inicialización

```
char unti[12] = { 'u', 'n', 't', 'i', 'v', 'e', 'r', 's', 'i', 'd', 'a', 'd' };
```

- Formato de lectura "%s" para leer/escibir arrays de caracteres

Funciones para entradas-salidas de cadenas interactivas.

### ENTRADA

- char \*gets (char \*s), lee la siguiente linea de entrada de la stdin (pantalla) y la almacena en el arreglo s que es su argumento. reemplaza el carácter nuevo linea con '\0'.

### SALIDA

- int puts (const char \*s), escribe la cadena s en la stdout (pantalla) y manda un carácter de nuevo linea a la pantalla.

## Resumen

`gets` → convierte el retorno de linea en un caracter nulo

El get en teoría se usa: `int cad[100]; get(cad)`

`puts` → convierte el caracter nulo en un retorno de linea

`fr puts " " " " " " " ; puts(cad);`.

## Funciones de manejo de cadenas:

`<string.h>` Con `cad1` y `cad2` arreglos de caracteres:

\* `int strlen(cad1)` Retorna la longitud de `cad1`

\* `int strcmp(cad1, cad2)` compara `cad1` con `cad2`, carácter a carácter.

Si (`cad1[i] < cad2[i]`) Entonces

retorna entero  $< 0$ .

Si (`cad1[i] = cad2[i]`) .. ..

retorna 0

Si (`cad1[i] > cad2[i]`) .. ..

retorna entero  $> 0$ .

\* `int strcpy(cad1, cad2)` copia lo `cad2` a `cad1`, incluyendo el terminalor "`\0`". Retorna `cad1`

\* `int strcmp(cad1, cad2, n)` compara hasta  $n$  caracteres de los `cad1` con `cad2` sin diferencias Maxus de mayus

\* `char *strncpy(cad1, cad2, n)` copia hasta  $n$  caracteres de `cad2` en `cad1`. Retorna `cad1`. Rellena con "`\0`" si `cad2` tiene menos de  $n$  caracteres.

\* `char *strcat(cad1, cad2)` concatena lo `cad2` al final de `cad1`. Retorna `cad1`.

Ejemplo en fotocopia. VER!

## A Modo de Resumen:

Los arreglos, como una unidad, no admiten:

- Asignación directa entre ellos.
- Operaciones aritméticas directas
- Comparaciones directas.
- Devolución como valor de devolución de una función

void invertir (int nn, int arr [TAMA]) {

```
int tempo, i;  
  
for (i=0; i < nn/2; i++)  
{ tempo = arr [i];  
arr [i] = arr [(nn-1)-i];  
arr [(nn-1)-i] = tempo;
```

}

}.

## PUNTEROS

Declarativa      nombre ← Acceso directo vía nombre  
                       espacio de almoc // a dress.

vble      dir. de memo

a      0000 004F

b      0000 F02A.      Definiciones.

Ac indirecto  
                       vía direc.

- Un puntero es una variable que contiene la dirección de memoria de otra variable

- Un puntero es una variable que representa la posición (en vez del valor) de otro dato.

- Un puntero es una variable que contiene una dirección de memoria de otro valor.

Los punteros son una herramienta muy poderosa.

Punteros y direcciones.

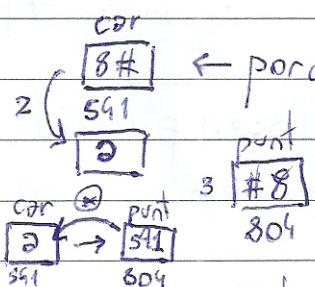
Puntero = pointer = ↑ (simbolización)

1- declaro variable 'car'

2- car ← 'a'

3- declaro una var. puntero 'punt'

4- relaciono 'punt' y 'car'



② ligado con 'a' a través  
                       de su dirección de memoria.

La variable puntero ocupa como mínimo 2 bytes.

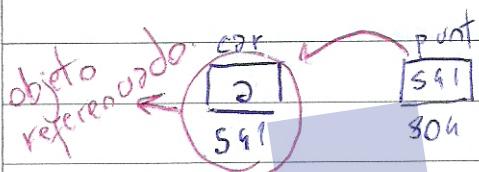
Las variables de tipo puntero se declaran como cualquier otra variable en C. Tiene 3 elementos:

- tipo base
  - operador de puntero: \*
  - identificador o nombre.
- } tipo base \* identificador  
Sintaxis.

Ejemplo:

char\* punt;

Una variable puntero puede referenciar, apuntar o referir.



pdfelement

Una variable puntero también puede cambiar su contenido

### Operadores puntero

\* , & son los dos operadores puntero

\* : operador de indirección o referencia  
& : operador de referencia. → permite tomar la dir del operando

por ejemplo el scanf ("%d", &num); guarda el dato en la dirección de memoria de num.

El \* sirve para declarar la variable puntero o acceder al contenido.

El & permite tomar la dirección del operando (la variable num, por ejemplo)

8

## Ejemplos

`int a, b;``int *p;`

• → saca la dirección de memoria de a.

`p = &a;`

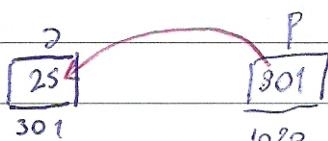
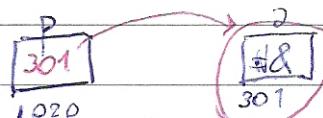
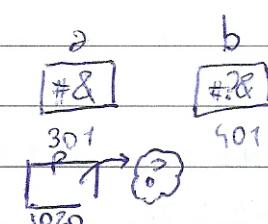
Así se le asigna un valor a la variable puntero

`a = 25;`

• nombre \ dirección acceso

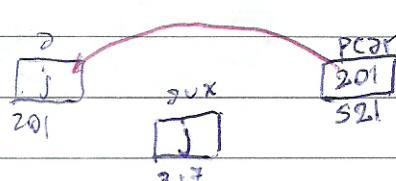
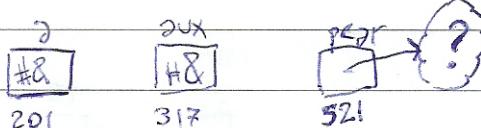
- `printf("%d", a);`

- `printf("%d", *p);` Así se imprime el contenido de la variable a apuntada por p.



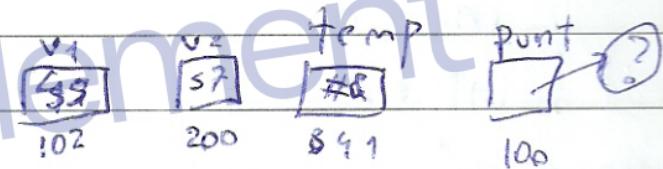
Para escribir direcciones de memoria, es decir el CONTENIDO de p y no de a, se usa %u:  
`printf("%u", p);`

## Ejemplo II

`char a, aux; *pcar;``a = 'j';``pcar = &a;``aux = *pcar;`

Ejemplo de un intercambio con puntero.

int v1, v2, temp, \*punt;  
v1 = 45; v2 = 57;



punt = &v1;

temp = \*punt; punt = &v2;

v2 = temp;

v1 = \*punt.



6

## Punteros y enteros

Punteros y enteros no son intercambiables.

`pcar = char ; X . // Esto es solo válido en el caso NULL  
entero int *pent = NULL;`

`int a;`

`pent = &a; warning`

Los archivos se pueden agregar y usar solo con un uso de punteros y de NULL

## Operaciones con puntero

▲ Asignaciones

▲ Aritméticos

▲ Comparaciones

pdfelement

### Asignaciones

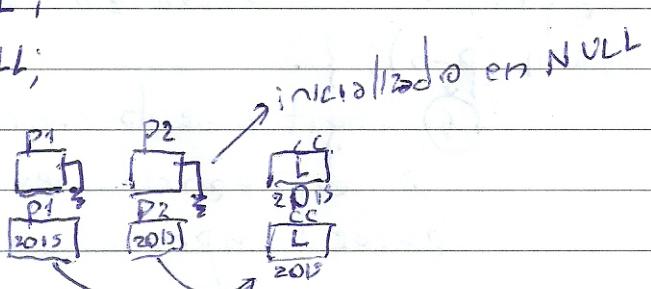
Entre punteros: `char *p1 = NULL;`

`char *p2 = NULL;`

`char cc = 'L';`

`p1 = &cc;`

`p1 = p2;`



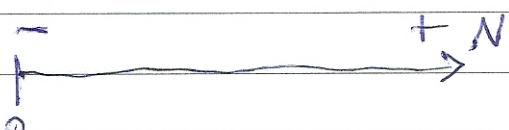
### Aritméticos

`+ -`

En la suma entre punteros se mueve la dirección de memoria a la derecha

En la resta `--` `--` `--` `--` `--` `--`

izquierda



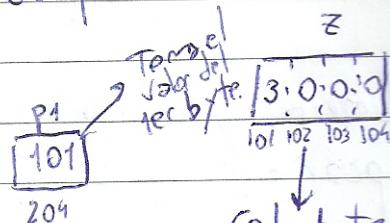
+ - } muy relacionadas c/tipo base del puntero.

`int z = 3000; int *p1 = NULL;`

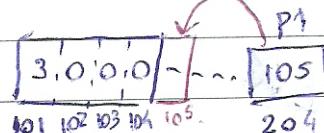
`p1 = &z;`

:

`p1 = p1 + 1; // p1++`



cada byte  
está en una  
dirección  
distinta.

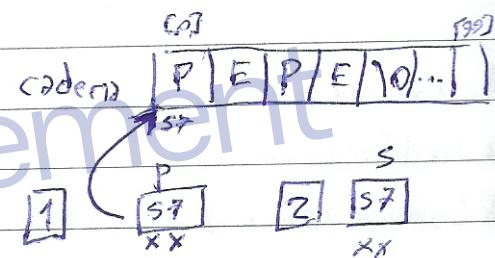


- ⊕ Se desplaza tantos bytes
- ⊖ como tenga el tipo base

Se pueden apuntar arreglos.

`int mstrlen (char cadena[80])`

```
{ char *p = cadena; ①
  char *s = cadena; ② }
```



`while (*p != '\0')`

`p++;` → Avanza de a 1 byte porque `char=1byte`.  
`return (p-s);`.

④ longitud de la cadena.

la diferencia devuelta es un VALOR que se  
puede imprimir como %d, %u, %p etc.

Recordar que los arreglos son bidireccionales. Si se modifiican en una función ya se modifican en el Main, por lo que por lo general las funciones son vacías.

(10)

## punteros y Arreglos

`float *pflo = NULL;`

`float arre = {90.4, 3.14, 5.8, 9.3};`

`pflo = &arre[0]; //versito`

~~`pflo = &arre[0];`~~, sirve pero es demasiado rebuscado

~~`pflo = &arre[0];`~~ ESTO ESTA' MAL.  
 $\rightarrow$  direcc(puntero).

`arre  $\rightarrow$  arre[0]`

`arre + 1  $\rightarrow$  arre[1]`

`arre + i  $\rightarrow$  arre[i]`

`float *pflo = NULL;`

`(pflo + i)`

`arre + i`

`arre[i]`

`pflo[i]`

Ejemplo

```
int a[MAX], suma = 0, i, (*p = NULL);  

//int *p = NULL;
```

```
for (p = a; p < a[MAX]; p++)  

    suma = suma + (*p);
```

## Punteros, arreglos y funciones.

Ejemplo: función cambiar a minú:

- 1) void cambiar (char cadena [MAX], int n);
- 2) void cambiar (char \*punt);

Se puede asignar a un puntero otro puntero.  
por ejemplo si `cad` es un arreglo (el versito) por el  
versito la palabra `cad` es un puntero.  
entonces

`char *p;`

`p = cad;`

### Tipos de parámetros de funciones

- Por valor: Esta forma de acceder a los datos, si se lo usa para parámetros de funciones no cambia el valor original de la variable
- Por referencia: Si hace cambios permanentes en el valor original. Por ejemplo como los arreglos

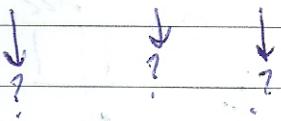
#### Por valor

La variable transferida es copiada en esa dirección de memoria, y todas las transacciones que se hacen sobre la variable transferida se reflejan en esa dirección de memoria, por lo que se dice que la función trabaja sobre una copia de la variable.

## Funciones con Parámetros por Referencia



$\text{pro} \leftarrow M1 + M2 + \text{const}$



Incrementos  $M1 \leftarrow M1 * 1.03$

$M2 \leftarrow M2 * 1.05$

Función: calculo

E:  $M1, M2, \text{real} > 0$

S: p.promedio : real

-  $M1 \leftarrow \text{idem}$

$M2 \leftarrow \dots$

RETORNAR  $(M1, M2)$ .

float calculo (float  $* M1, float  $* M2,$$

{ float nue, aux1, aux2;

$\text{aux1} = * M1;$

$\text{aux2} = * M2;$

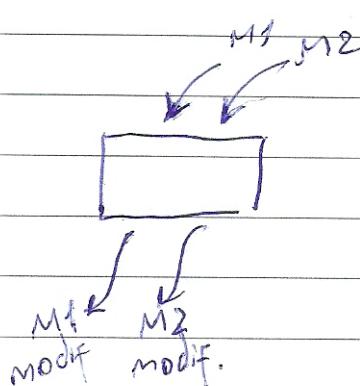
$\text{aux1} = \text{aux1} * 1.03; \text{aux2} = \text{aux2} * 1.05;$

$* M1 = \text{aux1};$

$* M2 = \text{aux2};$

$nue = \text{aux1} + \text{aux2} + \text{cte.}$

Return (nue); }



(12)

## Unidad 10. tipos de datos derivados: Estructura.

Que es una estructura?

- Una estructura es una colección de uno o más tipos de variables, agrupadas bajo un mismo nombre para un manejo conveniente

Ejemplo de Estructuras.

Las estructuras permiten asociar informaciones, o lo que es lo mismo, permiten que un grupo de variables relacionadas sean tratadas como una unidad en lugar de entidades separadas.

Ejemplo:

- Nombre → char nom[MAX]
- Apellido → char apel[MAX]
- Edad → int edad
- Fecha de nacimiento →
- Peso → float peso
- Estado Civil → char estado (S o C)

Definición I:

Definición II: Una estructura es una forma de agrupar un conjunto de datos de distinto tipo bajo un mismo nombre identificador

Definición III:

## Características generales de las estructuras

- Tienen un nombre
- Elementos de distintos tipos llamados campos o miembros
- Acceso a los componentes vía <sup>nombre</sup>~~indice~~ o puntero
- Acceso a los componentes individualmente ó como un todo (vía ~~indice~~ o puntero)

¿Cómo crear una Estructura?

se habrá de usar la palabra reservada struct  
struct [nombre]{

```
    tipo; var1;
    tipo; var2;
    .
    .
    tipo; var3;
    .
    tipo; varn;
};
```

- (nombre: opcional, se llama rotulo ó etiqueta de la estructura).
- (tipo; var; : son los campos o miembros de la estructura.  
Son variables que se declaran con su tipo asociado, que puede ser cualquier tipo válido.)

Declaración de variables de tipo estructurado

Una vez definido el nuevo tipo de datos, se estará en condiciones de declarar variables asociadas a él, por ejemplo:

struct persona{

char nom[MAX];

char ape[MAX];

int edad

float peso

char sexo

}

struct persona PACIENTE;

struct persona CLIENTE;

struct PERSONA, CLIENTE;

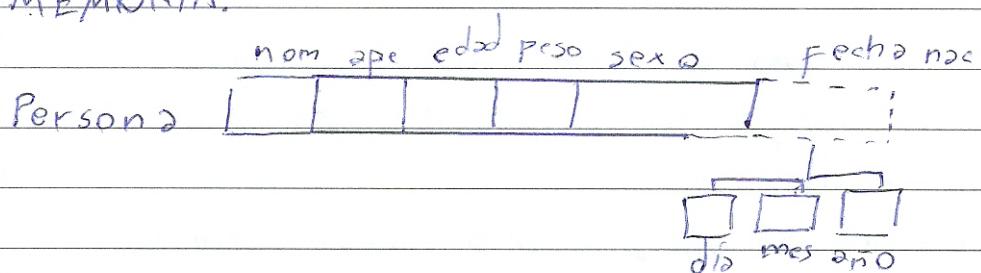
variables de tipo

struct persona

(tipo estructuradas)

(13)

La declaración de las variables "PACIENTE" y "CLIENTE" asociadas al tipo persona si implica una reserva de memoria.



struct persona { — }

struct fecha { int dia, mes, año }  
};

Inicialización de una variable asociada al tipo estructura.

- Se puede inicializar una variable asociada al tipo estructura en su declaración, ellos se harán siguiendo su declaración con una lista de inicializadores, cada uno con una expresión constante para sus miembros

struct PUNTO {  
 int puntoX;  
 int puntoY;  
};

int main()

{

struct PUNTO punto1 = {7, 5};

...

return 0;

}.

Como acceder a los campos o miembros de una estructura

- El acceso a los miembros o campos de una variable asociada al tipo estructura se hace a través del OPERADOR PUNTO
- La forma de uso del mismo es:  
Nombre\_de\_la\_variable.nombre\_del\_campo.

// acceso Directo (vía nombre)

struct persona {

};

int main()

{

    struct persona PACIENTE;

    ...

    printf("%d", PACIENTE.edad);

    printf("%f", PACIENTE.peso);

    printf("%c", PACIENTE.sexo);

    return 0; } // gets y puts.

Operaciones

- copiar como una unidad
- asignar entre ellas
- acceder y trabajar con sus campos.
- sumar estructuras
- usar como argumentos de funciones; completas y por partes
- Tomar su dirección (&)
- ser devueltas por funciones.

NO SE PUEDEN COMPARAR!

(14)

Como cargar datos en una variable estructurada  
struct persona

{  
= -  
};

struct persona individuo; //OPERADOR PUNTO!  
gets(individuo.apellido);  
scanf("%d", &individuo.edad)

Se pueden asignar:

struct persona PACIENTE1, PACIENTE3;

PACIENTE3 = PACIENTE1; //asignación

Estructuras anidadas.

Se puede usar una estructura adentro de otra estructura.  
En estos casos para acceder a un campo de la estructura interior habrá que usar tantos puntos como niveles se tengan.

persona.fecha.nsc. año.

A tener en cuenta:

- La declaración del tipo estructurado no implica una reserva de memoria
- Una declaración de una estructura se corresponde con la definición de un nuevo tipo de dato.
- Plantilla o molde para definir variables con la forma de la plantilla.
- La declaración de las variables estructuradas si implica

una reserva de memoria.

- Para usar el operador punto, se debe hacer a través del nombre de la variable, acceso directo.
- El nombre de un campo no se puede repetir en una definición de estructura.
- Puede haber definiciones de estructuras distintas con miembros de igual nombre
- Una definición de tipo estructura es una definición de tipo jerárquico, esto es, las estructuras pueden anidarse  
los structs se pueden hacer arreglos.  
y cada componente struct personas estudiantes [50]; es un struct.

[ ] ] . . . . ]  
[0]

struct personas estudiantes [50].

```
{ nom[MAX];
  ape[MAX];
}
```

variables estructuras y funciones

Las operaciones entre ambas pueden ser:

Se pueden hacer funciones que devuelven estructuras 

Usar campos de una variable estructura como argumentos  
de funciones

.

Ejemplo.

```
float promedio (float n1, float n2, float n3);  
struct Alumno {  
    char ape[100];  
    char nom[100];  
    int edad;  
    float nota1;  
    float nota2;  
    float nota3;  
};  
int main(void){  
    struct Alumno alum;  
    float prom  
    ...  
    prom = promedio (alum.nota1, alum.nota2, alum.nota3);  
    ...  
}
```

Usar variables estructuradas completas.  
como argumentos de funciones.

Ejemplo:

~~float promedio (struct Alumno alum);~~

Funciones que devuelven estructuras

struct Alumno {

char ape[100];

char nom[100];

int edad;

float nota1;

float nota2;

float nota3;

}

Struct alumno cargo Datos (void).

{ struct Alumno alum;

puts ("Ingresar apellido del alumno: ");

gets (alum.apellido);

fflush (stdin);

fflush (stdout);

puts ("Edad: ");

scanf ("%d", &alum.edad);

return (alum);

(16)

m.marguetti@herrera.unt.edu.ar

Estructuras - Punteros - Funciones.

```
struct alumno { char nom_apel [80];
                short n1, n2, n3; };
```

```
struct alumno ALU, *punt=NULL;
*punt=NULL
```

```
void cargar_datos (struct alumno *estud);
```

```
main ()
```

```
{
```

```
    cargar_datos (&ALU);
}
```

1 FORMA

```
void cargar_datos (struct alumno *estud) {
    struct alumno A1;
    gets (A1.nom_apel);
    scanf ("%d %d %d") &(A1.n1), &(A1.n2), &(A1.n3);
    *estud = A1;
}
```

Cuando se utiliza un puntero estructura, para acceder a los campos se utiliza el operador  $\rightarrow$

En el ejemplo anterior

estud  $\rightarrow$  nom\_apel; o de otra forma  
 $(*punt)$ . DNI.