Universidad Católica del Norte

ESCUELA DE INGENIERÍA UCN COQUIMBO

Introducción a la programación en C

Juan Bekios Calfa
http://jbekios.ucn.cl
juan.bekios@ucn.cl

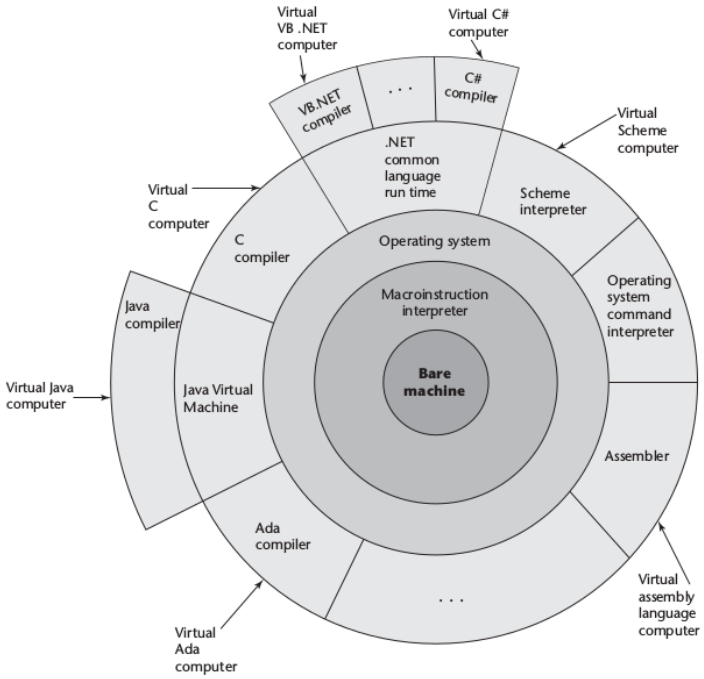
Universidad Católica del Norte

ESCUELA DE INGENIERÍA UCN COQUIMBO

Temario

- Conceptos Básicos
  - Arquitectura de un computador
  - Intérpretes y Compilador
- Introducción a C
- Punteros en C
- String
- Funciones
- Struct

Vistas por capas de un computador



Intérpretes y Compiladores de Lenguajes de Programación

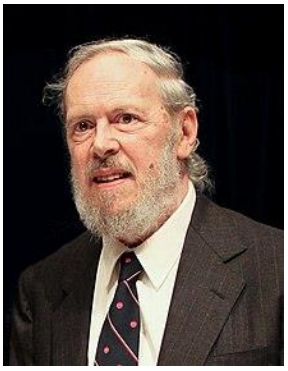
La mayoría de las implementaciones de un lenguaje de programación (ej. Java y C) necesitan dos etapas para ejecutar un programa: ***compilar e interpretar***

Compilación: *Durante esta etapa se verifica errores de "tipos" y realiza optimizaciones*

Interpretación: *Aquí, se interpreta el programa usando una "maquina virtual" o directamente el sistema operativo*

Introducción a C

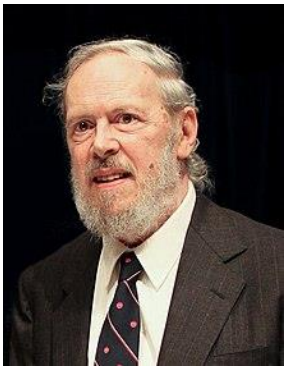
- Creado en 1969-1973, luego tuvo una extensión llamada C++, la cual soporta Programación Orientada a Objetos (POO) y otras características. Java usa POO también.
- Muchos lenguajes están inspirados (al menos en sintaxis) en C . Por ejemplo, Java, PHP, PYTHON, C#, Perl, Ruby, etc.
- Hoy en día C, se usa principalmente para crear aplicaciones de bajo nivel (muy abajo en las capas de abstracciones de un SO), donde la eficiencia es un requisito importante
- C soporta “if”, ciclos, arrays, y como las mayorías de los lenguajes Turing-complete



Dennis Ritchie
09/09/1941-12/10/2011

Introducción a C

- Creado en 1969-1973, luego tuvo una extensión llamada C++, la cual soporta Programación Orientada a Objetos (POO) y otras características. Java usa POO también.
- Muchos lenguajes están inspirados (al menos en sintaxis) en C . Por ejemplo, Java, PHP, PYTHON, C#, Perl, Ruby, etc.
- Hoy en día C, se usa principalmente para crear aplicaciones de bajo nivel (muy abajo en las capas de abstracciones de un SO), donde la eficiencia es un requisito importante
- C soporta “if”, ciclos, arrays, y como las mayorías de los lenguajes Turing-complete



Dennis Ritchie
09/09/1941-12/10/2011
Desarrollador C

Bjarne Stroustrup
Desarrollador C++

Conociendo C

```c
#include<stdio.h>

int  main() {//aqui hay un comentario
  printf("Hello World\n");
  return 0;
}
```

¿Por qué “f” en “printf”?

C en un ejemplo básico (1)

```c
#include<stdio.h>

 int main() {
       //aquí hay un comentario
       int a = 2;
       float b = a;
       printf(“El valor de es %f”, b);

      return 0;
}
```

https://repl.it/@PaulLeger/clase12

C en un ejemplo básico (2)

```
#include<stdio.h>

 int main() {
       //aquí hay un comentario
       int a = 2;
       printf(“Dame un número:”);
       scanf(“%i”,&a);
       for (int i = 0; i < a ; ++i) {
         printf (“El número es %i\n”, i);
      }
      return 0;
}
```

https://repl.it/@PaulLeger/clase13

Resumen del formato para mostrar

- %i o %d -> integer
- %f -> float
- %lf -> double
- %c -> char
- %s -> string
- %p -> pointer

Ejercicio

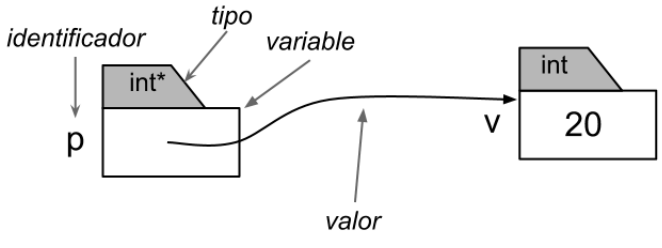
Implemente un programa que dado un número ingresado por el usuario, diga si es primo o no

Use: http://replt.it

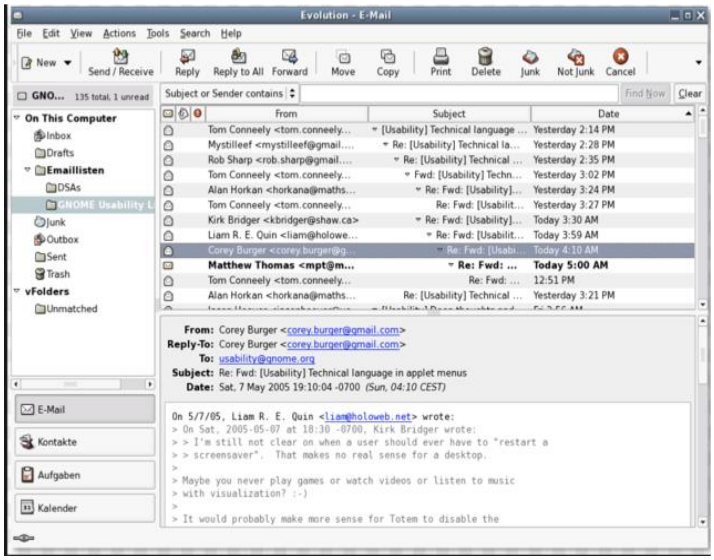
Punteros: Característica distinguida de C

- Permite administración directa de la memoria (*first-class memory*) a través de ¡PUNTEROS!

- Lo anterior lo vuelve muy eficiente, es decir, programas que se ejecutan muy rápidos. Sin embargo, también se vuelve extremadamente peligroso

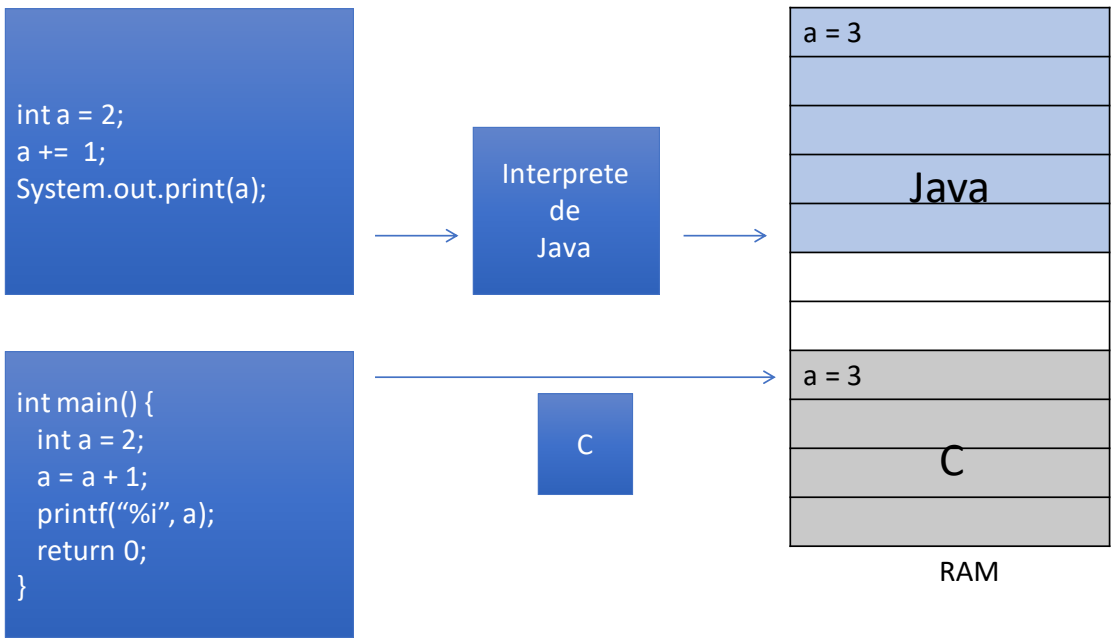


Software Evolution E-Mail



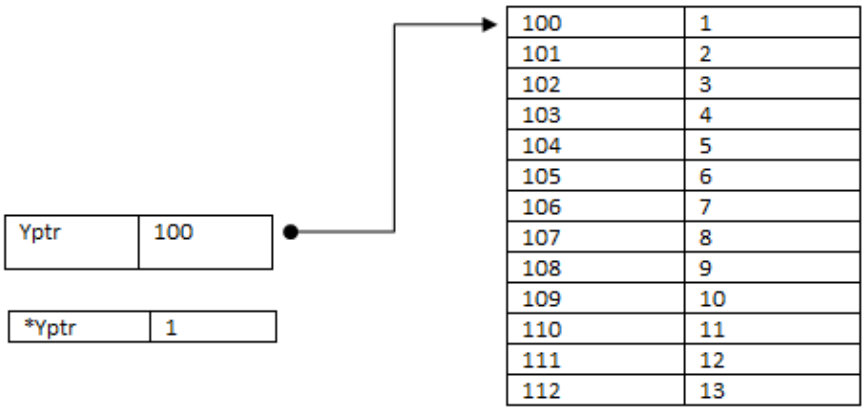
- Evolution mail es un cliente de e-mail para Gnome manager implementado en C
- Cada vez que reparaban un error, agregaban tres errores más
- Hoy, Ubuntu está reemplazando C por Python y JavaScript (**NOTA:** la parte interna del sistema operativo sigue siendo C)

Uso de Memoria



Administración de Memoria con Punteros

Suponga una variable Yptr, cuya dirección es 100 y su valor 1



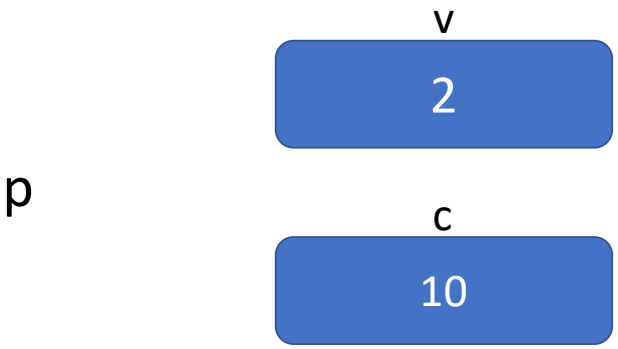
Punteros y Valores

```
int v = 2, c = 10;
int *p;
```

1)
```
p = &v;
*p = 50;
```

2)
```
p = &v;
*p = c;
```

3)
```
p = &c;
*p = &v;
```



Ejemplos de Punteros (1)

https://replit.com/@jbekios/clase01-Puntero01#main.c

```c
#include<stdio.h>

int main() {
   int a = 20, b = 4;
   int *c;

   c = &b;
   *c = 5;
   printf("%i",*c);
   b = 7;
   printf("%i",*c);
}
```

Ejemplos de Punteros (2)

https://replit.com/@jbekios/clase01-Puntero02#main.c

```c
#include<stdio.h>

int sumar1(int a, int b) {
     int  c = a + b;
     b = a;
     return c;
}

int sumar2(int *a, int *b) {
     int  c = *a + *b;
     *b = *a;
     return c;
}

int main() {
      int a = 2, b = 4;
     // int c1 = sumar1(a, b);
      int c2 = sumar2(&a, &b);
      printf("%i", b);
      return 0;
}
```

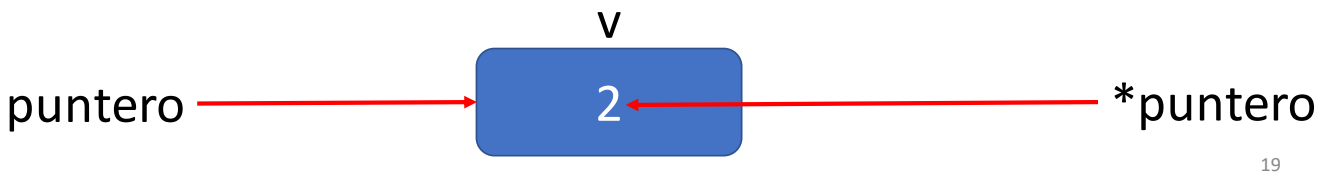
NO OLVIDAR

Puntero: Variable cuyo valor que contiene es una dirección de memoria (referencia)

int *puntero; *//Declaración de un puntero **ENTERO***

puntero = &v; *//puntero contiene la dirección de memoria donde esta v (la cual debe ser entera)*

***puntero = c;** //puntero almacena el VALOR ENTERO de c en la dirección de memoria que apunta PUNTERO*



Cosas que mantener en mente (1)

```
int v = 2;
int *p;

p = &v;      //puntero ”p” apunta a la direccion de v
int c = *p;  //puntero “p” accede el valor que direcciona

c = *p + *p;

c = p; // no es correcto, pero es solo un warning
```

Ejemplos de Punteros (2)

```c
#include<stdio.h>

int main() {
  int v = 2;
  int *p;

  p = &v;
  printf("1. %i \n",(*p));

  v = 5;
  printf("2. %i \n",(*p));

  (*p)++;
  printf("3. %i \n",(v));

  p++;
  printf("4. %i \n",(v));

  p++;
  printf("5. %i \n",(*p));

  p+= 1000;

  printf("6. %i \n",(*p));

  return 0;
}
```

- Operadores de aritméticos con punteros es controlar la dirección de memoria
- Una prohibición de acceso puede gatillarse si no se controla bien
- ¿Qué pasa si …?

  int *mypointer = &variableNormal;
  char password[10];
  mypointer++; //y llego a password!!??

https://repl.it/@PaulLeger/clase15

Accediendo a otras variables en C

```c
#include <stdio.h>

int main() {
  int v = 2, password = 30;
  int *p;

  p = &v;
  p = p - 1;
  printf("%i \n",*p);
  return 0;
}
```

¿Qué muestra?

https://repl.it/@PaulLeger/clase16

Cosas que mantener en mente (2)

```
int v[5] ;
int *p;

p = &v[0]; //apuntar a la primera posición del arreglo
p = v; //igual que el anterior
```

RECORDAR: ¡v aquí es un arreglo!
Por eso es válido esa expresión

Ejemplos de Punteros (3)

```c
#include<stdio.h>

int main() {
  int v[5];
  v[0] = 10;
  v[1] = 20;
  v[2] = 30;
  v[3] = 40;
  v[4] = 50;

  int *p = &v[0]; //v;

  for (int i = 0; i < 4; ++i) {
    printf("%i\n", p[i]);
  }

  return 0;
}
```

- P[i] podría también ser *(p++)
- Tener cuidado con el largo del arreglo

https://repl.it/@PaulLeger/clase17

Cosas que mantener en mente (3)

Un arreglo pasado a una función siempre pasa por puntero:

- ¿ “int f(int v[5])” == “int f(int *v)”?
Si

Esto significa que un arreglo puede ser modificado dentro de una función.
(NOTA: Al igual que Java)

Ejemplos de Punteros (4)

```c
#include<stdio.h>

//patch with "const"
int sumarVector(int* v) {
  int suma = 0;
  for (int i = 0; i < 4; ++i) {
      suma += v[i];
  }
  v[0] = 300000;
  return suma;
}


int main() {
  int v[5];
  v[0] = 10;
  v[1] = 20;
  v[2] = 30;
  v[3] = 40;
  v[4] = 50;

  int res = sumarVector(v);
  printf("res = %d\n", res);

  for (int i = 0; i < 4; ++i) {
    printf("%d\n", v[i]);
  }
  return 0;
```

- “const” evita que un puntero sea modificado dentro de una función

https://repl.it/@PaulLeger/clase18

Cosas que mantener en mente (4)

Un puntero sin dirección valida debería ser NULL

int *p = NULL;

int c = 1;
if (2==c) printf("hola");
else printf("chao");

o en una versión mejorada en C++
int *p = nullptr;

C sufre un problema de convertir los tipos de manera automáticas (ej. boolean pasa a entero)

¿El siguiente código funciona?

```
#include<stdio.h>

int main() {
  int *x;

  *x = 5;

  printf("x = %i \n", *x);

  return 0;
}
```

Aquí no apunta a algo desconocido

https://stackoverflow.com/questions/2235457/how-to-explain-undefined-behavior-to-know-it-all-newbies

Un posible “segmentation fault”

```c
#include<stdio.h>

int main() {
  int *x;
  *x = 5;
  printf("x = %i \n", *x);
  for( int i=0; i<1000; i++) { int* seg_fault_generator; *seg_fault_generator = 5; }

  return 0;
}
```

Aquí no apunta a algo desconocido (potencialmente inválido)

https://repl.it/@PaulLeger/clase18-1

Ejercicio

Implemente la función “swap”, la cual cambia los valores de dos variables.

NOTA: Sí, use punteros

String y sus librerías en C (C++ es diferente)


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

int main() {

  char s[5];
  strcpy(s,"hola"); //char s[5] = "hola";

  printf(" string1 = %s \n", s);

  s[0] = 't';

  printf(" string2 = %s \n", s);

  printf(" Son iguales: %d\n", strcmp(s, "tola"));

  return 0;
}
```

Un ***string*** es un arreglo de char

Se modifica un string como si fuera un arreglo

Existe azúcar sintáctico como *s[5] = "hola"*

Se puede usar con punteros, pero recuerde reservar memoria para el string … (más adelante con esto)

¿es válido escribir char *s ="hola"?

https://repl.it/@PaulLeger/clase19

Alternativas para escribir funciones

```
funcion
{
        --------
        --------
        --------
}


main()
{
        funcion()
        //llamado a función
}
```

```
extern funcion()  //declaración de la función

main ()
{
        extern funcion() //declaracion de la función

}

funcion
{
        ---------
        ---------
        ---------
}
```

```c
#include <stdio.h>

int sumarDosNumeros(int a, int b) {
  return a + b;
}

extern int restarDosNumeros(int a, int b);

int main (){

  printf ("Hola mundo\n");
  printf ("Hare unos calculos\n");
  int suma = sumarDosNumeros(5,6);
  int resta = restarDosNumeros(5,6);
  printf("La suma 5 con 6 es %i y su resta es %i \n",suma,resta);
  return 0;
}

int restarDosNumeros(int a, int b) {
  return a - b;
}
```

Funciones inline

Una declaración de función puede estar precedida del especificador ***inline***. Esto fuerza al compilador a sustituir el cuerpo del código para la función a la ubicación donde es llamada la función, mejorando su eficiencia

```
inline void funcion1(int a, int b){

…
…
…

}
```

Desventajas:
- Si la función es usada muchas veces, un librería o archivo se vuelve pesado para cargarse en memoria
- Si se usan muchas variables, puede ocurrir un sobre costo de números de variables
- Trashring Memory
- Si el “header (h)” tiene “inline”, se puede volver poco leible (más adelante encontrar un ejemplo de ello)

Constantes, tipos y declaraciones

```c
#include <stdio.h>
const int TAMANIO = 10;

float datos[TAMANIO];
char titulo[] = "Primer programa";

int main()
{
  extern float obtenerCuadrado (float *valor, int n);
  extern void ingresarValores (float *datos, int n);

  int cantidad;
  printf("%s", titulo);
  printf("\n\n¿Cuántos valores deben ser ingresados?");
  scanf("%i", &cantidad);
  ingresarValores (datos, cantidad);
  printf("\nLa suma de los cuadrados de los valores=%f \n",
    obtenerCuadrado(datos, cantidad));

  return 0;
}


float obtenerCuadrado (float valor [], int n)
{
  float sum = 0.0;
  for (int i = 0; i<n; i++)
    sum+=valor[i] *valor[i];
  return sum;
}


void ingresarValores (float[] valor, int n)
{
  for (int i=0; i<n; i++)
  {
    printf("\nIngrese número%d: ", i+1);
    scanf("%f", &valor[i]);
  }
}
```

- Definición de constantes
- Definición de variables globales

https://repl.it/@PaulLeger/clase110

Ejercicios propuestos

Escriba las siguientes funciones (en una hoja):

1. **void mostrarArreglo(int a[],int size);** // muestra un arreglo de tamaño size

2. **void invertirArreglo(int a[], int size);** // invierte (o revierte) los valores de un arreglo

3. **int esNumeroPerfecto(int n);** // retorna 1 si n es perfecto, en caso contrario retorna 0. Un número perfecto es un número natural que es igual a la suma de sus divisores exactos positivos. Ejemplos, 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14

4. **int isPalindrome(const char *s);** // Una palabra es palíndroma si esa palabra se lee de igual manera desde izquierda a derecha y derecha a izquierda. Esta función retorna 1 si es palíndroma o 0 si no lo es. Ejemplos: ana, reconocer, ala.

Declaración de un struct

Cada componente de la estructura denota una propiedad particular de la abstracción. La declaración denota una molde debido a que no representa a una instancia específica. Para declarar tipos de esta estructura:

```
struct Persona juan, pedro, karen;
```

En el ejemplo superior, se han declarado cinco Tipos de Datos Abstractos (TDA) distintos, cada uno de los cuales requiere una cierta cantidad de espacio en memoria. Todos los TDAs tienen las mismas propiedades

```
struct Persona {
  char nombre[100];
  int numeroSocial;
  float salario;
};
```

Ejemplo del uso de una estructura

¡¡¡¡ Una estructura no es un objeto !!!!!

Una estructura no tiene asociado un constructor o métodos

Similar a Java, el "." para usar al atributo

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

struct Persona {
  char nombre[100];
  int numeroSocial;
  float salario;
};

int main() {

  struct Persona juan;
  strcpy(juan.nombre, "Juan");
  juan.numeroSocial = 23;
  juan.salario = 12.3;

  printf("nombre: %s\n",juan.nombre);
  printf("numero Social: %d\n",juan.numeroSocial);
  printf("Salario: %f\n",juan.salario);

  return 0;
}
```

https://repl.it/@PaulLeger/Clase111

Ejemplo del uso de una estructura y Typedef

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

struct Persona {
  char nombre[100];
  int numeroSocial;
  float salario;
};

typedef struct Persona TipoPersona;

int main() {

  TipoPersona juan;
  strcpy(juan.nombre, "Juan");
  juan.numeroSocial = 23;
  juan.salario = 12.3;

  printf("nombre: %s\n",juan.nombre);
  printf("numero Social: %d\n",juan.numeroSocial);
  printf("Salario: %f\n",juan.salario);

  return 0;
}
```

Para ahorrase "struct TIPO NOMBRE_NUEVO_TIPO", es posible definir un tipo con typedef

typedef struct Persona TipoPersona;

También puede hacer:

typedef struct Persona Persona;

https://repl.it/@PaulLeger/clase112

Struct en funciones

```c
typedef struct Persona Persona;

void esAltoSalario(Persona p) {
  if (p.salario > 100) {
    printf("Es alto\n");
  } else  {
    printf("Es menor\n");
  }
}

void anonimizar1(Persona p) {
  strcpy(p.nombre,"anon");
}

void anonimizar2(Persona *p) {
  strcpy(p->nombre,"anon");
}

int main() {
  Persona juan;
  strcpy(juan.nombre, "Juan");
  juan.numeroSocial = 23;
  juan.salario = 12.3;

  printf("nombre: %s\n",juan.nombre);
  printf("numero Social: %d\n",juan.numeroSocial);
  printf("Salario: %f\n",juan.salario);

  esAltoSalario(juan);

  anonimizar1(juan);
  printf("nombre: %s\n",juan.nombre);

  anonimizar2(&juan);
  printf("nombre: %s\n",juan.nombre);

  return 0;
}
```

Las estructuras en funciones funcionan como los valores primitivos (ej. int, float), es decir, sus modificaciones no son visibles por fuera. NO ES COMO JAVA

Para modificar un atributo dentro de una función (COMO EN JAVA), se debe usar punteros.
El "." se intercambia por "->".

https://repl.it/@PaulLeger/clase112-1

Punteros en Struct

```c
void esAltoSalario(Persona p) {
  if (p.salario > 100) {
    printf("Es alto\n");
  } else  {
    printf("Es menor\n");
  }
}

void anonimizar1(Persona p) {
  strcpy(p.nombre,"anon");
}

void anonimizar2(Persona *p) {
  strcpy(p->nombre,"anon");
}

int main() {
  Persona juan;
  Persona *p;

  p = &juan;

  strcpy(p->nombre, "Juan");
  p->numeroSocial = 23;
  p->salario = 12.3;

  printf("nombre: %s\n",p->nombre);
  printf("numero Social: %d\n",p->numeroSocial);
  printf("Salario: %f\n",p->salario);

  esAltoSalario(juan);

  anonimizar1(juan);
  printf("nombre: %s\n",juan.nombre);

  anonimizar2(&juan);
  printf("nombre: %s\n",juan.nombre);

  return 0;
}
```

Los punteros en struct funciona de manera similar que las variables usadas en C

https://repl.it/@PaulLeger/clase113-1