UNIVERSIDAD DEL BÍO-BÍO

SISTEMAS OPERATIVOS

INTRODUCCIÓN AL LENGUAJE C

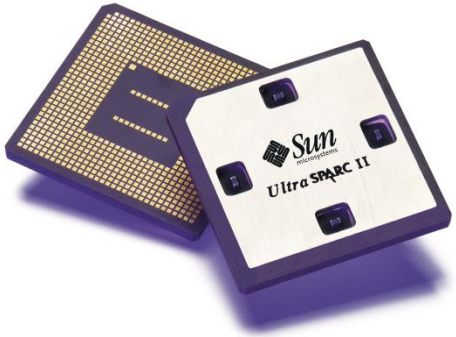
Fernando Santolaya F.
Ayudantía de Sistemas Operativos
Material compartido por Luis Gajardo D.

ESTADO ACTUAL DE LA TECNOLOGÍA DE PROCESADORES: RISC VS CISC

- **RISC**: Reduced Instruction Set Computer
  Reducido Conjunto de Instrucciones Computacionales

  Ej: SUN Sparc, MIPS.

  Power pc, play station 3, Wii, xbox 360



- **CISC**: Complex Instruction Set Computer
  Complejo Conjunto de Instrucciones Computacionales

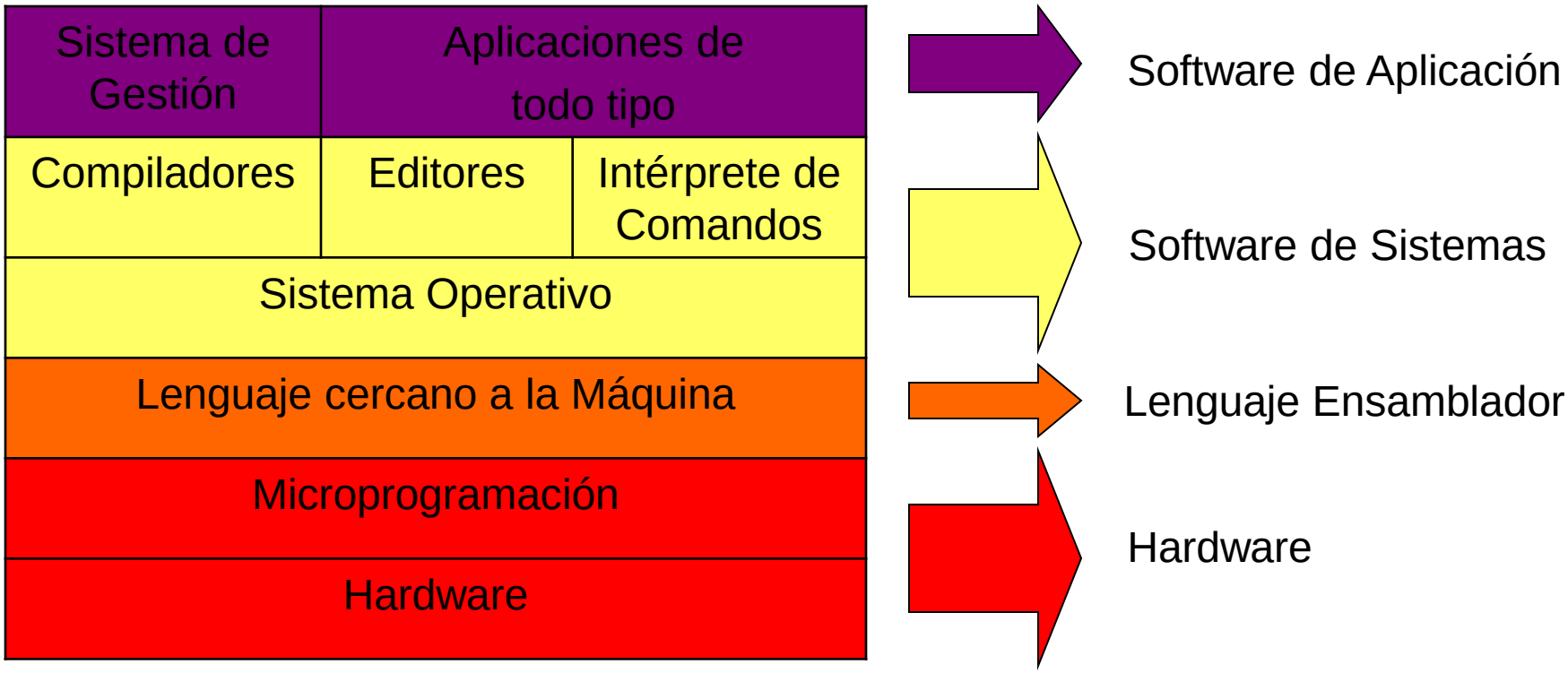
  Ej: Intel, AMD

RISC V/S CISC

	CISC (Complex Instruction Set Computer)	RISC (Reduced Instruction Set Computer)
1	Emphasis on hardware	Emphasis on software
2	Includes multi-clock complex instructions	Single-clock, reduced instruction only
3	Small code sizes	Typically larger code sizes
4	Many addressing modes	Few addressing modes.
5	An easy compiler design	A complex compiler design.
6	Pipelining does not function correctly here because of complexity in instructions.	Pipelining is not a major problem and this option speeds up the processors.

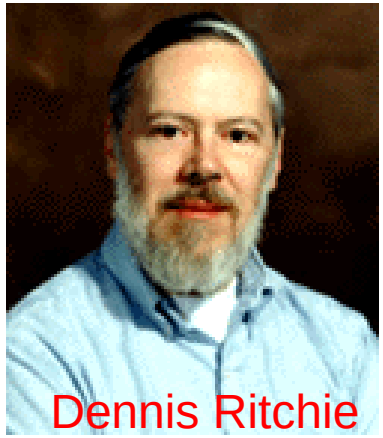
Revisar el siguiente link:
http://www.xataka.com/componentes-de-pc/cisc-frente-a-risc-una-batalla-en-blanco-y-negro

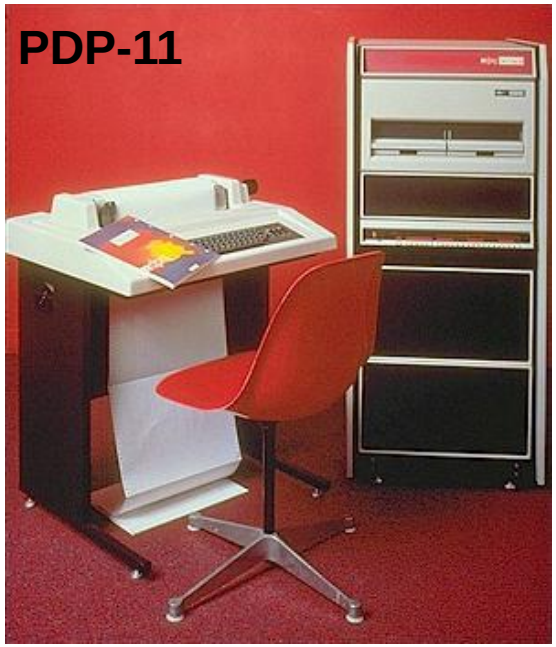
ESTRUCTURA DEL SOFTWARE DE UN SISTEMA

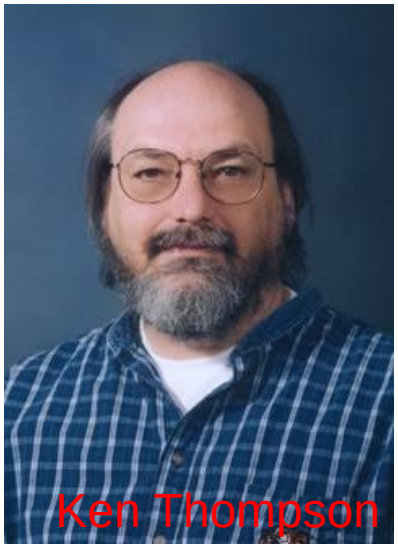


HISTORIA DEL LENGUAJE C

- C fue creado por **Dennis Ritchie** en 1972 en un computador DEC PDP-11, con sistema operativo UNIX.
- C es sucesor del antiguo lenguaje B, desarrollado por **Ken Thompson**.
- La primera versión de C fue el estándar utilizado para la construcción del verdadero UNIX actual “UNIX System V”.
- Su rápido éxito ocasiona la aparición de muchas variantes y problemas de compatibilidad.


Dennis Ritchie




Ken Thompson

HISTORIA DEL LENGUAJE C

- Existe un **estándar ISO** de 1986 denominado **ANSI C**. En teoría, un lenguaje 100% ANSI C sería portable entre plataformas y/o arquitecturas pero en la práctica esto no es siempre cierto.
  (ANSI: American National Standars Institute).

- En 1989 se adopta otro estándar con lo cual aparecen los primeros compiladores para C.

- En 1999 se adoptó el último estándar conocido como C99 con algunas mejoras e ideas de C++.

- Actualmente cohexisten ambos estándares (C89 y C99) pero se está migrando a este último.

CARACTERISTICAS

- C es un lenguaje estructurado que permite trabajar a alto y bajo nivel, esto permite mayor potencia y flexibilidad a cambio de menos abstracción.

- No es un lenguaje fuertemente tipado, lo que significa que se permite **casi cualquier conversión de tipos**.

- **No se llevan a cabo comprobaciones de errores en tiempo de ejecución**. (Ej.: no se comprueban los índices de los arreglos).

- Tiene un reducido número de palabras clave: 32 en C89 y 37 en C99.

- Los compiladores de C, en general, producen programas ejecutables muy eficientes y portables.

cc: + eficiente en compilación
- eficiente en ejecución

gcc: - eficiente en compilación
+ eficiente en ejecución

CARACTERISTICAS

Obs: La optimización depende de la arquitectura de la CPU. Por ejemplo, el código generado en un procesador RISC es aproximadamente un 30% del generado en un CISC.

- C dispone de una biblioteca estándar de funciones.

En resumen:

C es un lenguaje muy flexible, muy potente, muy popular pero que **NO PROTEGE AL PROGRAMADOR DE SUS ERRORES**.

DIFERENCIAS ENTRE C Y C++

- C++ es un lenguaje para programación orientada a objetos que toma como base el lenguaje C.

- Existen algunas diferencias sintácticas entre ambos.

  Casi todas las diferencias son del tipo:
  "que tiene C++ que no tiene C".

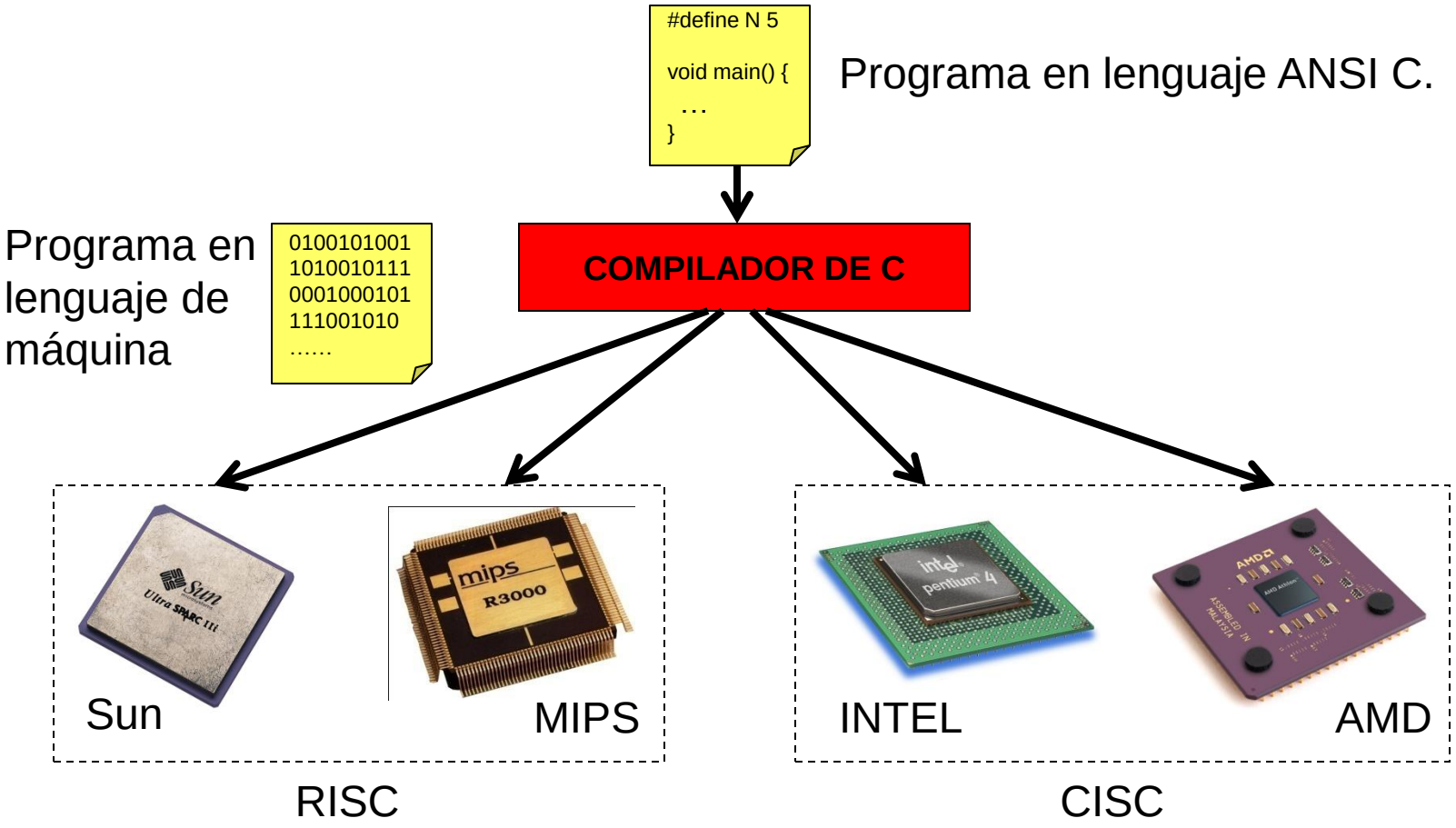
  - En C++ es obligatorio usar prototipos de funciones, en C no siempre es necesario.

  - C++ utiliza "//" también para los comentarios.

  - etc.

PORTABILIDAD DE C

El compilador de C genera código ejecutable multiplataforma

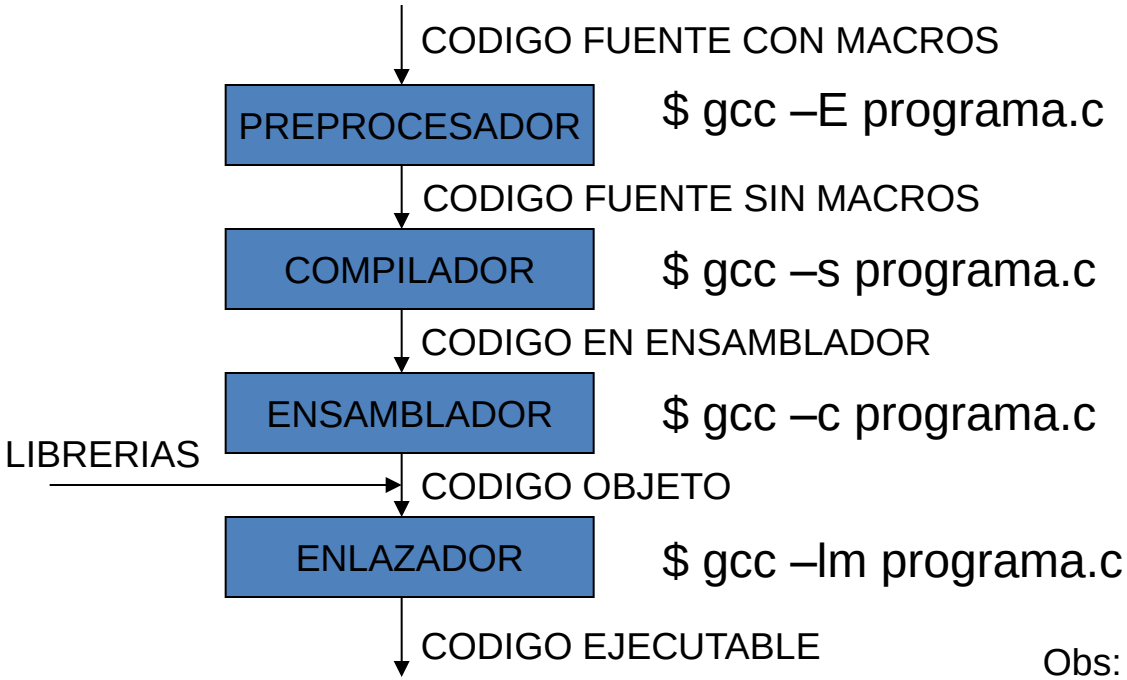


COMPILACIÓN

MODELO DE COMPILACION DE C

- Existen muchos compiladores de C. El cc es el compilador estándar de Sun Microsystems. El compilador GNU para C es gcc, el cual es bastante popular y existe para varias plataformas.

- En general todos los compiladores de C operan de forma estándar y comparten muchas opciones comunes en la línea de comandos:



Obs: lm indica librería matemática.

MODELO DE COMPILACION DE C

PREPROCESADOR

- Acepta el código fuente como entrada y es responzable de:
  - Quitar los comentarios
  - Interpretar las directivas del preprocesador (#)

#include

incluye el contenido del archivo nombrado. Usualmente estos archivos se llaman “cabecera” (header).

Ej: #include <stdio.h> biblioteca estándar de E/S
#include <math.h> biblioteca matemática

#define

Define un nombre simbólico o constante. Permite crear “macros”.

```
Ej: #define  TAM_MAX_ARREGLO  100
    #define  TRUE  1
    #define  FALSE  !TRUE
    #define  MAYOR(X,Y)  X>Y
```

MODELO DE COMPILACION DE C

COMPILADOR

- El compilador traduce código fuente en código ensamblador, el código fuente lo recibe del preprocesador.

- El compilador de C es uno de los más eficientes que existen, el código generado es compacto y consume pocos recursos.

- El código ensamblador puede ser obtenido directamente con la opción “–s”, luego el archivo puede ser abierto con un editor ASCII y puede ser leído y modificado normalmente.

MODELO DE COMPILACION DE C

ENSAMBLADOR

- Permite tomar el código en lenguaje ensamblador y generar un código denominado “objeto”.

- El código objeto es binario y posee todas las instrucciones assembler traducidas a cero y unos, salvo las instrucciones que forman parte de las librerías utilizadas en el programa.

MODELO DE COMPILACION DE C

ENLAZADOR

- El enlazador es el encargado de tomar el código objeto y agregarle el código relativo a las librerías utilizadas en el programa, el código generado en esta etapa es llamado ejecutable.

- Hay dos tipos principales de enlazado (linkado):

  - **Enlazado Estático:** Los binarios de las librerías se añaden a nuestros binarios compilados.

  - **Enlazado Dinámico:** Esto no añade los binarios de las librerías a nuestro binario, si no que hará que nuestro programa cargue en memoria la librería en el momento que la necesite. Esto permite que nuestro programa no crezca demasiado en tamaño.

COMPILADOR gcc

- gcc (GNU Compiler Collection) es un compilador rápido, muy flexible y riguroso con el estándar de C ANSI.

- Puede funcionar como compilador cruzado (generando código para plataformas distintas a la de desarrollo)

- gcc no proporciona un entorno integrado de programación, es decir no es un IDE.

  Uso: $ gcc [opciones] [archivos …]

COMPILADOR gcc

- Algunas opciones más usadas son:

--help:	mostrar la ayuda de gcc.
-o [archivo]:	permite colocar un nombre al archivo ejecutable generado en la compilación. (por defecto es a.out)
-Wall:	muestra todos los errores y advertencias al compilar.
-g:	incluye en el archivo ejecutable información necesaria para usar posteriormente un depurador (por ejemplo gdb)
-O [nivel]:	permite optimizar el código (el nivel va de 0 a 3). Obs: no use la opción –O cuando use la opción –g.
-E:	sólo realiza la fase del preprocesador, no compila ni ensambla.
-S:	Genera sólo el archivo ensamblador del programa
-c:	Genera sólo código objeto

COMPILADOR gcc

- Al compilar un archivo fuente con gcc, el compilador nos informa con dos tipos de mensajes diferentes:

  Error: fallo al analizar el código C que impide la generación de un ejecutable final.

  Warning: advertencia del compilador al analizar el código C que no impide la generación del ejecutable final.

  Ejemplo: *holamundo.c*

```
#include <stdio.h>

int main(void) {
  printf("Hola mundo\n");
}
```

COMPILADOR gcc

Para compilar este programa usamos:

```
$gcc holamundo.c
holamundo.c: In function ‘main’:
holamundo.c:6:warning: control reaches end of non-void function
```

A pesar del warning gcc compila y genera un ejecutable. La causa de esta advertencia es debida a la falta de la **sentencia return** al final del programa.

```
#include <stdio.h>

int main(void) {
  printf(“Hola mundo\n”);
  return 0;
}
```

Podemos utilizar el comando ls para ver el directorio, debería aparecer un archivo llamado “a.out”.

PUNTEROS

INTRODUCCION

- La memoria RAM del computador puede ser vista como un arreglo de bytes.

- Dentro de la memoria cada dato ocupa un número determinado de bytes.

  Ejemplo: un char → 1 byte
  un int → 2 o 4 bytes (depende de la arquitectura o procesador)

- A cada byte se puede acceder por su dirección.

- Un puntero es una variable que contiene una dirección de memoria de un byte.

- Normalmente esa dirección apunta a otra variable en memoria.

- Una declaración de un puntero consiste en un tipo base, un * (asterisco) y el nombre de la variable.

  declaración: ***tipo *nombre_variable;***
  *float *pf;*

- El tipo del puntero define el tipo de variables a las que puede apuntar.

INTRODUCCION

- Existen dos operadores para trabajar con los punteros:
  - El operador de dirección (&) devuelve la dirección de memoria de una variable.
  - El operador de indirección (*) devuelve el contenido de la dirección apuntada por el puntero.
- Cuando se declara un puntero, este apunta a una dirección desconocida.
- Cuando se asigna un cero a un puntero, este no apunta a ninguna posición de memoria. La constante simbólica NULL, definida en <stdio.h> tiene el valor cero y representa el puntero nulo.

```
int *p = NULL;
```

- Cuando se declara un puntero con el tipo void, este puede apuntar a una variable de cualquier tipo (ya que no está ligado a ningún tipo específico).

declaración: *void *pointer;* (también se les llama **punteros genéricos**)

INTRODUCCION

- El principal uso de los punteros es para permitir la utilización de memoria dinámica y además para permitir la utilización de argumentos pasados a una función por referencia.

Ejemplo: argumentos pasados por referencia

```
void swap(int *x, int *y) {
  int aux;
  aux= *x;
  *x= *y;
  *y= aux;
}
```

```
int a=100, b=200;
swap(&a, &b);
```

ASIGNACION DE PUNTEROS

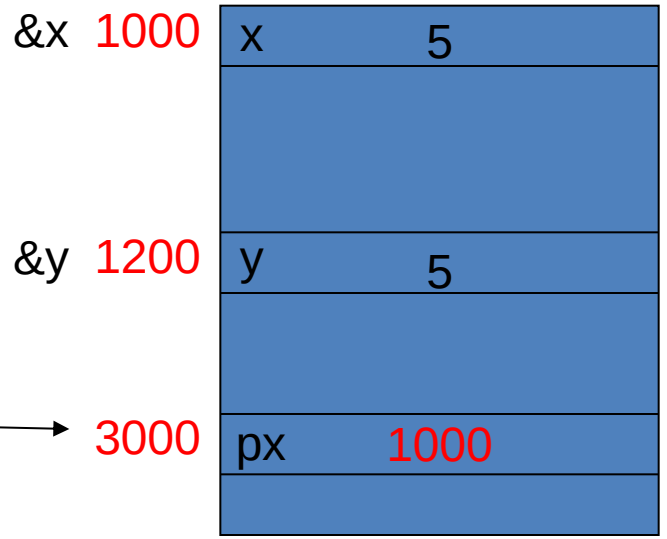
- Ejemplo1:

```
…
int x;
int *p1, *p2;
p1= &x;
p2= p1;  /*tanto p1 como p2 apuntan a x*/
…
```

- Ejemplo2:

```
#include <stdio.h>

void main() {
  int x, y;
  int *px;
  x = 5;
  px = &x;
  y = *px;
  printf("x = %d \n", x);   → 5
  printf("y = %d \n", y);   → 5
  printf("*px = %d \n", *px); → 5
}
```



UN PUNTERO NO RESERVA MEMORIA

- Ejemplo: `int *p;` ERROR
  `*p= 5;` es incorrecto porque no sabemos a que dirección apunta p

El siguiente fragmento de código si es correcto:

```
char *p;
char letra;
letra= 'a';
p= &letra;
*p= 'b';     /* letra también vale 'b' */
letra= 'c'; /* *p también vale 'c' */
```

ARITMETICA DE PUNTEROS

- Existen sólo dos operaciones aritméticas que se pueden usar con punteros: suma (+, ++ pre y post) o resta (-, - pre y post)

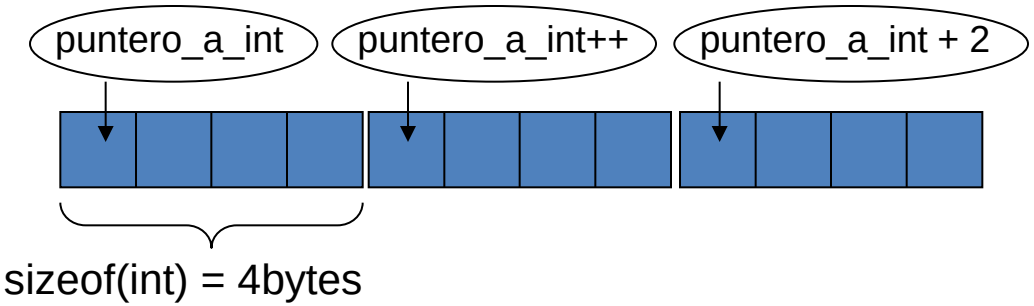
- Es posible usar operaciones de relación (comparaciones: <, >, <=, >=, ==, !=)

- No se puede multiplicar o dividir punteros y no se le puede sumar o restar un valor de tipo float o double a un puntero.

- Cada vez que se incrementa un puntero, apunta a la posición de memoria del siguiente elemento de su tipo base.

- Se debe tener en cuenta que “puntero + x” apuntará a la dirección de puntero sumándole x veces el espacio ocupado por un elemento del tipo al que apunta.

Ejemplo1:

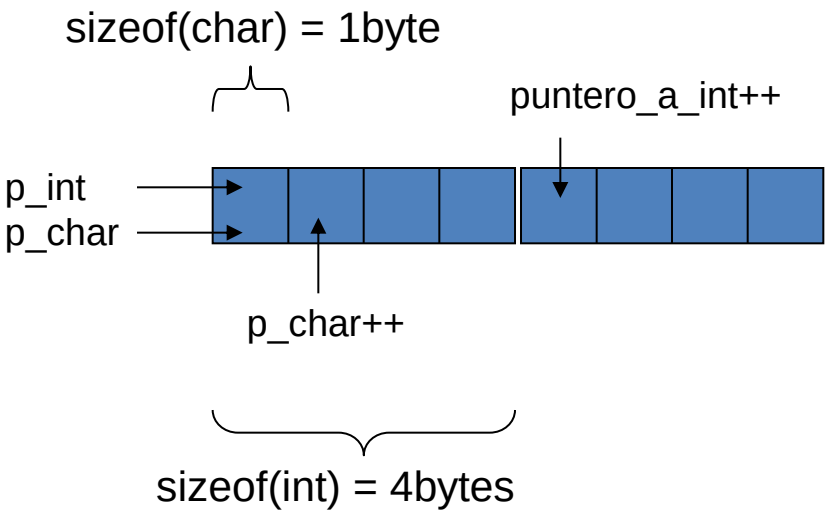


Asumiendo que un entero ocupa 4 bytes

ARITMETICA DE PUNTEROS

- Ejemplo2:

```
int *p_int;
char *p_char;
p_int = p_char;
p_int = p_int++;
p_char= p_char++;
```



- p_int se desplaza 4 bytes
- p_char se desplaza un solo byte

- El tipo void no soporta aritmética de punteros, ya que no se sabe cuántos bytes ocupa el dato que referencia.

PUNTEROS A ARREGLO

- Existe una estrecha relación entre los punteros y los arreglos. Considere el siguiente código:

```
char cad[8], *pc;
pc= cad;
```

Aquí *pc* ha sido asignado a la dirección del primer elemento del arreglo *cad*.

Para acceder al quinto elemento de *cad* se escribe : cad[4] o (pc+4)

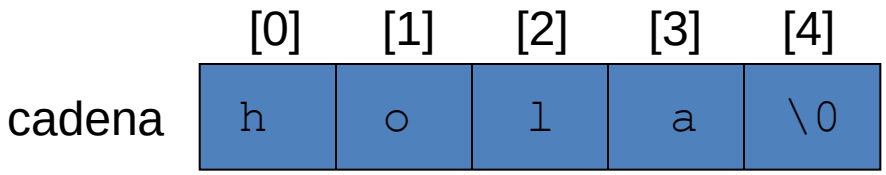
- Un nombre de arreglo sin índice devuelve la dirección del comienzo del arreglo, que es el primer elemento, es decir: cad == &cad[0]

- El compilador traduce la notación de arreglos en notación de punteros.

PUNTEROS A ARREGLO

- Ejemplo3:

```
char cadena[5];
char *puntero;
puntero = &cadena[0]; /*puntero a la cadena cero*/
*puntero= 'h';        /*cadena[0]='h'; */
*(puntero+1)= 'o';    /*cadena[1]='o'; */
*(puntero+2)='l';     /*cadena[2]='l'; */
*(puntero+3)='a';     /*cadena[3]='a'; */
*(puntero+4)='\0';    /*cadena[4]='\0';*/
```



ARREGLOS DE PUNTEROS

- Los punteros pueden estructurarse en arreglos como cualquier otro tipo de datos.

- La declaración, por ejemplo para un arreglo de 10 punteros a enteros es la siguiente:

*int *x[10];*	distinto de	*int (*x) [10];*
Arreglo x con 10 punteros a enteros		Puntero x a un arreglo de 10 enteros

- Para asignar la dirección de una variable entera llamada *var* al tercer elemento del arreglo de punteros se escribe:

```
x[2]= &var;
```

- Se puede recuperar el contenido de *var* de la siguiente manera:

```
n= *x[2];
```

INDIRECCION MULTIPLE

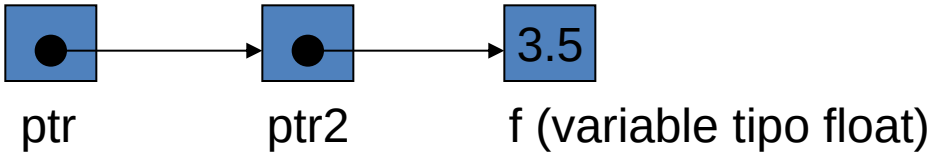
- Se puede hacer que un puntero apunte a otro puntero el cual apunta a alguna variable (o dato). Esta situación se denomina indirección múltiple o puntero a punteros.

- Una variable que es un puntero a puntero tiene que declararse como tal. Esto se hace colocando un asterisco adicional en frente del nombre de la variable.

- Ejemplo:

*float **ptr;*

Obs: pueden haber más de dos niveles de punteros.



Dónde *ptr* es un puntero a un puntero (*ptr2*) a una variable (*f*).

FUNCIONES DE ASIGNACION DINAMICA DE MEMORIA

- Los punteros proporcionan el soporte necesario para el poderoso sistema de asignación dinámica de memoria de C.

- La asignación dinámica es la forma en la que un programa puede obtener memoria mientras se ejecuta.

- El centro del sistema de asignación de memoria dinámica está compuesto por las funciones *malloc()*, que asigna memoria y free(), que la libera.

- El prototipo de *malloc()* es:

  *void *malloc(int size);*

También se puede:

```
p=(struct auto) malloc(sizeof(struct auto));
```

- Tras una llamada exitosa *malloc()* devuelve un puntero (genérico) al primer byte de memoria reservada. Si no hay suficiente memoria libre para satisfacer la petición de *malloc()* se devuelve NULL (cero).

- El siguiente ejemplo reserva memoria para 50 enteros.

```
int *p;
p= (int *) malloc(50 * sizeof(int));
```

PUNTEROS A ESTRUCTURAS

- Funciona de manera similar que el resto de las variables.

- Ejemplo:

```
struct punto {
  float x;
  float y;
};

void main() {
  struct punto punto_1;
  struct punto *punto_2;
  punto_1.x= 2.0;
  punto_1.y= 4.0;
  punto_2= &(punto_1);
  printf("...
  ...
}
```

- En una variable de tipo puntero a estructura, los miembros se acceden con "→"

PUNTEROS A ESTRUCTURAS

- Ejemplo:

```
void leer_punto(struct punto *p);
Void imprimir_punto(struct punto p);

void main() {
  struct punto *p1;
  p1= (struct punto*) malloc (sizeof(struct punto));
  leer_punto(p1);
  imprimir_punto(*p1);
  free(p1);
}

void leer_punto(struct punto *p) {
  printf("x= ");
  scanf("%f", &(p->x));
  printf("y= ");
  scanf("%f", &(p->y));
}

void imprimir_punto(struct punto p) {
  printf("x= %f \n",p.x);
  printf("y= %f \n",p.y);
}
```

PUNTEROS A FUNCIONES

- Los punteros a funciones son útiles en casos específicos.

- El formato de la declaración es el siguiente:

  tipo (*identificador) (parámetros);

  *char (*f) (int, int);*

  f es un puntero a una función que devuelve un char y recibe dos enteros como argumentos.

- Otros ejemplos:

  *void (*f)();*
  *void (*f)(int);*
  *void *(*f)(int *, char *);*

Ejercicios en clase

- 1.- Dado un vector de 10 elementos ={1, 2, 3, 4, 4, 7, 8, 9, 5, 4}, escribir un programa en C (haciendo uso de puntero) que muestre las direcciones de memoria de cada elemento del vector.
- Genere un programa en c, con punteros, que concatene dos arreglos en uno. Puede pedir los datos por pantalla.