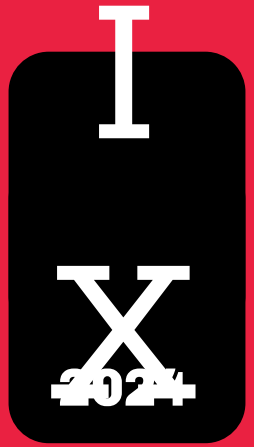
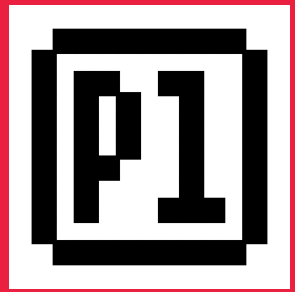
Punteros II





UNRN
Universidad Nacional
de **Río Negro**

rXX

Punteros a función

Cuestiones a tener frescas

1 Funciones

Cuestiones a tener frescas

2

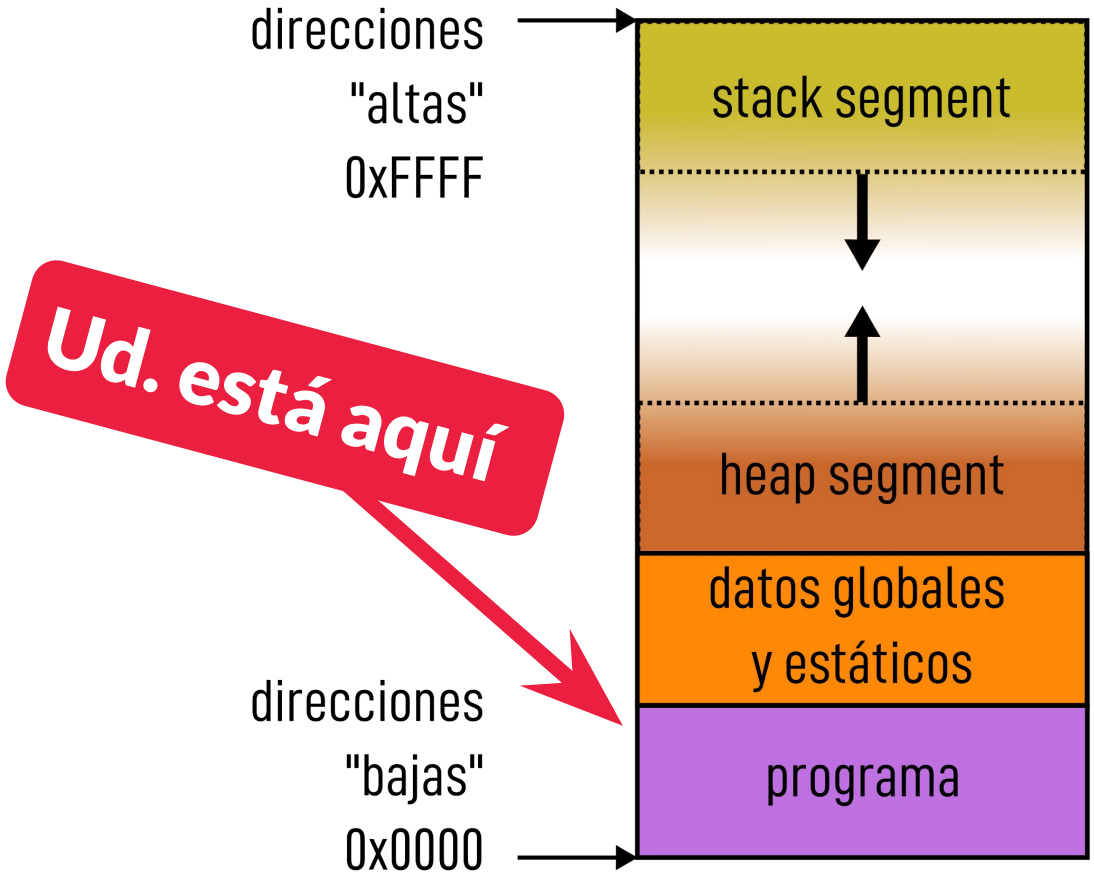
Punteros

Cuestiones a tener frescas

3

El mapa de la memoria

Código

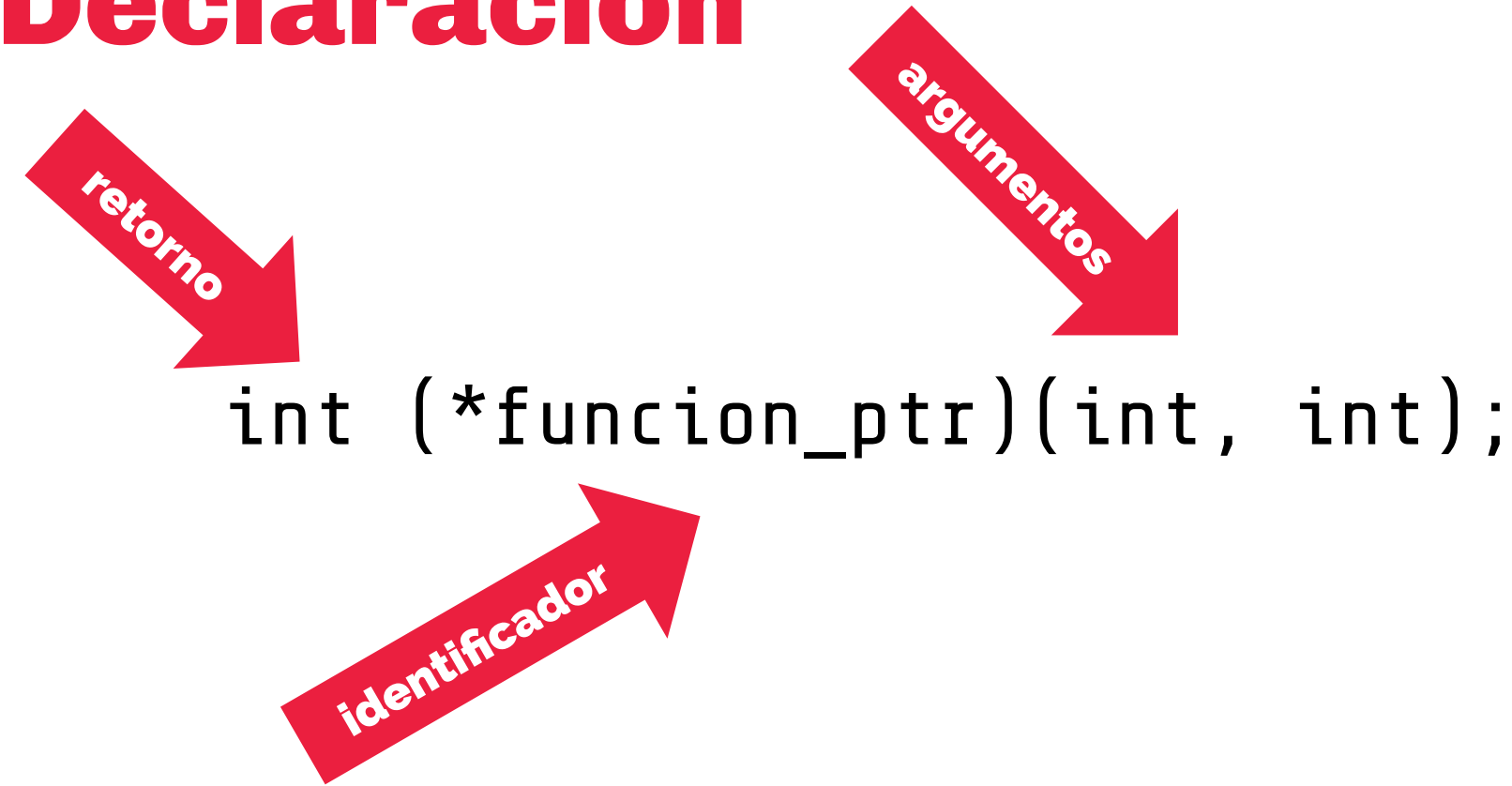

direcciones
"altas"
0xFFFF
stack segment
heap segment
datos globales
y estáticos
programa
direcciones
"bajas"
0x0000
Ud. está aquí

Están las presentaciones a mano para repasar


¿Preguntas?

¿Qué es un puntero a función?

Declaración



Dada esta función

```
int suma(int a, int b)
{
    return a + b;
}
```

Declaración y asignación

```
int (*funcion_ptr)(int, int);
funcion_ptr = &sumar;
```

Uso de suma via el puntero

```
int resultado = funcion_ptr(2, 3);
```

Un ejemplo completo

```c
#include <stdio.h>

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

int main() {
    int (*func_ptr)(int, int) = &sumar;
    int resultado = func_ptr(2, 3);
    printf("Resultado: %d\n", resultado);
    return 0;
}
```


¿Preguntas?

Pueden ir como argumentos

Esta función aplica func_ptr a todos los elementos

```
int operar(int largo, int *arreglo, int (*func_ptr)(int));
```

O retornar un puntero a función

¡Función selectora retorna un puntero a función!

```
int (*selectora(int argumento)) (int, int);
```

```c
int suma(int a, int b) {
    return a + b;
}

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

```c
int (*selectora(int opcion)) (int, int) {
    if (opcion == SUMA) {
        return &suma;
    } else {
        return &resta;
    }
}
```

Pero podemos usar `typedef` para simplificar

¡Simplificando!

Un puntero a función que recibe dos números y da uno de retorno

```
int (*func_ptr)(int, int);
```

Que como retorno queda:

```
int (*selectora(int opcion)) (int, int) {
```

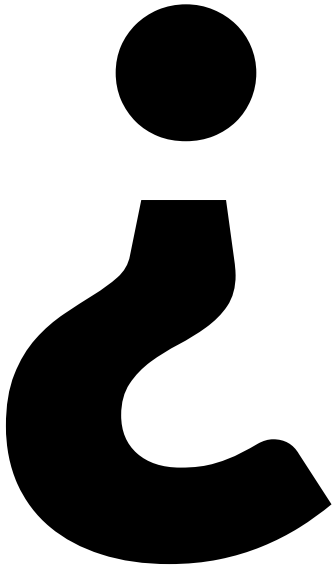
¡Hasta es entendible!

```
typedef int (*f_operacion_t)(int, int);

f_operacion_t seleccionarOperacion(int opcion) {
    if (opcion == 1) {
        return suma;
    } else {
        return resta;
    }
}
```

O como argumento

```
typedef int (*f_operacion_t)(int, int);

long aplicar(int *arreglo, int largo, f_operacion_t funcion);
```


¿Preguntas?



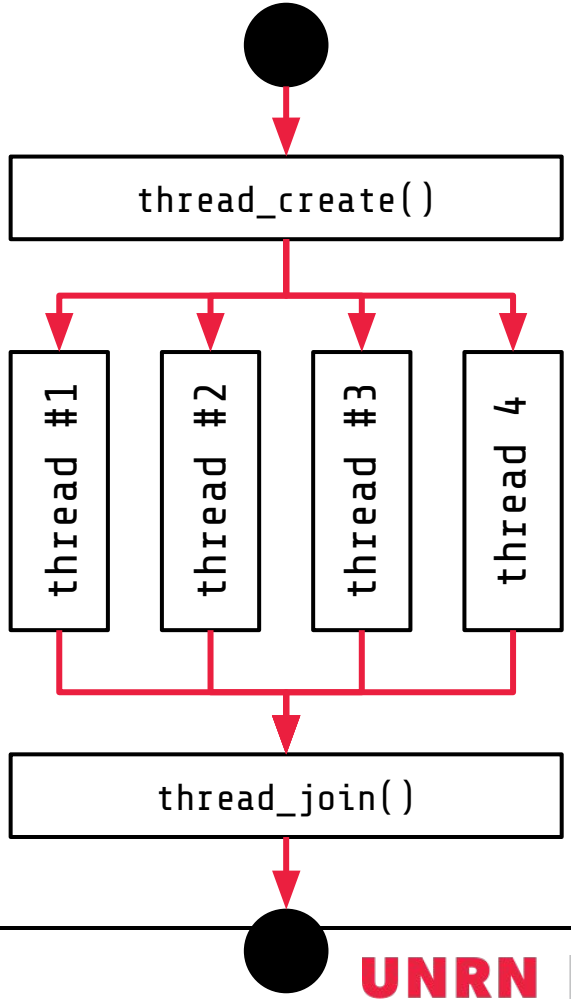
Para que se usan

1

Ejecución paralela con Hilos

Ejemplo ultra simple

```
int main() {
    pthread_t tid;
    Pthread_create(&tid, NULL, &hilo, NULL);
    Pthread_join(tid, NULL);
    ... //do some other work
    exit(0);
}

void *hilo(void *argumentos) {
    …// en paralelo!
}
```



Como diagrama de flujo

Como estrategia no es la única

Pero coordinar el trabajo es *supercomplicado*


¿Preguntas?

2

Datos + Código

Lo podemos combinar con estructuras

```
typedef int (*f_comparador_t)(void *, void *);

typedef struct{
    void* arreglo;
    size_t ancho;
    int largo;
    f_comparador_t comparador;
} t_arreglo;
```

Se convierte y obtiene el resultado

```
int comparador_enteros(void *a, void *b) {
    int int_a = *(int*)a;
    int int_b = *(int*)b;
    return *int_a - *int_b;
}
```

Inicialización

```
int cantidad_elementos = 5;

t_arreglo arr;
arr.largo = cantidad_elementos;
arr.ancho = sizeof(int);
arr.arreglo = malloc(arr.largo * arr.ancho);
arr.comparador = &comparador_enteros;
```

Cargamos el arreglo

```
int* numeros = (int*)arr.arreglo;
for (int i = 0; i < cantidad_elementos; i++) {
    numeros[i] = i + 1;
}
```

arr.arreglo => {1, 2, 3, 4, 5}

Y hacemos la búsqueda

```
int objetivo = 3;
int* encontrado = (int*)buscar(&arr, &objetivo);
```

Búsqueda genérica

```
void* buscar(t_arreglo* arr, void* buscado) {
    void* retorno = NULL;
    for (int i = 0; i < arr->largo; i++) {
        void* actual = ((char*)arr->arreglo)+(i * arr->ancho);
        if (arr->comparador(actual, buscado) == 0) {
            retorno = actual;
        }
    }
    return retorno;
}
```

Lo podemos combinar con estructuras

```
typedef int (*f_comparador_t)(void *, void *);
typedef void* (*f_busqueda_t)(arreglo_t*, void*);

typedef struct{
    void* arreglo;
    size_t ancho;
    int largo;
    f_comparacion_t comparador;
    f_busqueda_t buscador;
} t_arreglo;
```

La creación

```
arreglo_t crear_arreglo(int cantidad_elementos,
                        size_t tam_elemento,
                        f_comparador_t comparacion) {

    arreglo_t nuevo;
    nuevo.arreglo = malloc(cantidad_elementos * tam_elemento);
    nuevo.largo = cantidad_elementos;
    nuevo.ancho = tam_elemento;
    nuevo.comparador = comparacion;
    nuevo.buscador = &busqueda;
    return nuevo;
}
```

Orientación a objetos a los cascotazos

```
arr.buscador(arr, &objetivo);
```


¿Preguntas?

3

Algoritmos genéricos

Quicksort

en.cppreference.com/w/c/algorithm/qsort

```
void qsort( void* ptr, size_t count, size_t size,
            int (*comp)(const void*, const void*) );
```

Dado este arreglo

```
int numeros[LARGO] = {50, 23, 15, 51, 10, 20, 100, 30};
qsort(numeros, LARGO, sizeof(int), &compara);
```


https://en.cppreference.com/w/c/algorithm/qsort

Y la comparación se encarga del `cast`

```
int compara(const void *_a, const void *_b) {
    int *a = (int *) _a;
    int *b = (int *) _b;
    int retorno = 0;
    if (a < b){
        retorno = -1;
    } else {
        retorno = 1;
    }
    return retorno;
}
```

Búsqueda binaria

en.cppreference.com/w/c/algorithm/bsearch

```
void* bsearch( const void *key, const void *arr, size_t count, size_t size,
               int (*comp)(const void*, const void*) );
```


¿Preguntas?

4

Respuesta a eventos

Señales

`signal.h`

Esta función recibe un número y un puntero a función


```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int)
```

Y devuelve el mismo puntero a función

Con `typedef`s es *mucho* más entendible

```
typedef void (*f_handler_t)(int);

f_handler_t* signal(int signal, f_handler_t handler);
```

Dado este código

```
void handler(int);

int main () {
   signal(SIGINT, handler);

   while(1) {
      printf("A mimir por un segundo...\n");
      sleep(1);
   }
   return 0;
}
```

y el Handler

```
void handler(int signum) {
    printf("Recibimos un %d, chaucito...\n", signum);
    exit(1);
}
```

Algunas de las señales

```
SIGABRT   El programa reventó
SIGFPE    Excepción con número de punto flotante
SIGINT    Solicitud de interrupción (se puede ignorar)
SIGSEGV   Violación de segmento
SIGTERM   Solicitud de terminación (no se puede ignorar)
SIGUSR1   Señal de interpretación libre*
```

https://en.cppreference.com/w/c/program/SIG_types

Pueden venir de:

La terminal

Ctrl-C es SIGINT (interrumpir)

```
A mimir por un segundo...
A mimir por un segundo...
A mimir por un segundo...
A mimir por un segundo...
^CRecibimos un 2, adios
```

El sistema operativo

Pero también la podemos enviar manualmente

```
$> kill -SIGINT 386945
```

Y obtenemos el mismo resultado

Nuestro propio programa

Nos mandamos una señal a nosotros mismos

```
int raise( int sig );

…

raise(SIGTERM);
```

Interrupciones de Hardware*

No hay una forma estandarizada

Las señales se usan también para coordinar trabajo entre procesos

{

Esto es asíncrono*

}


¿Preguntas?


BUENO, MI TRABAJO HA TERMINADO
¿Y LA DOCUMENTACIÓN?
imgflip.com


¿Preguntas?

Hasta la próxima

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro