UNIVERSIDAD
NACIONAL DE
SAN MARTÍN

Programación UNSAM
Memoria dinámica

David López

v. 2020

Recapitulando: memoria y direcciones

- La memoria se divide en celdas de 1 byte (8 bits)
- Cada celda tiene una dirección única
- Una variable ocupa 1 o más celdas
- Todas las variables tienen asociada una dirección, que es la dirección de la primer celda que ocupan
- El operador & devuelve la dirección de memoria de una variable

Recapitulando: punteros

- Puntero:
  - Tipo especial de variable que almacena una dirección de memoria
  - Se dice que apunta a una variable si su contenido es la dirección de esa variable
  - Se debe declarar indicando a qué tipo de datos va a apuntar (int *, float * , struct xx *, void *, etc.). Por ejemplo:

```
int *p1;
float *p2;
struct producto *p3;
```

Recapitulando: indirección

- El operador * permite acceder al valor apuntado por un puntero
  - Siempre y cuando el puntero contenga una dirección válida

Recapitulando: punteros y vectores

- Los nombres de vectores y matrices son punteros
- Su nombre apunta al primer elemento
- Se pueden usar indistintamente los operadores * y [ ] para acceder a un elemento
- Aritmética de punteros: sumar o restar un entero a los punteros para moverse apuntando a los diferentes elementos de un vector o matriz

Recapitulando: punteros y vectores/matrices

- Ejemplo: si v es un vector e i es un entero entonces:

Esto	Es lo mismo que
`v`	`&v[0]`
`v+i`	`&v[i]`
`*(v+i)`	`v[i]`
`*v`	`v[0]`

- Las matrices son vectores de vectores
  - Ej: si m es una matriz de 2 dimensiones, entontes m[i] es un vector correspondiente a la fila i

Valor NULL

- Es una constante entera (#define) predefinida
- Si un puntero tiene asignado ese valor significa que en ese momento no contiene ninguna dirección válida
- En ese caso no se puede usar el operador *

```c
#include <stdio.h>

int main(void) {
   int *p;

   p = NULL;
   printf ("%d", *p); //Genera error en tiempo de ejecucion
   return 0;
}
```

Punteros genéricos (void*)

- El tipo void* puede apuntar a cualquier tipo de dato pero no se puede usar el operador * (indirección).
  - Se debe convertir el puntero void* a un tipo concreto para poder recuperar el dato apuntado
    - Esto se llama casting y se hace anteponiendo el tipo de puntero al cual convertir entre paréntesis (ver ejemplo en sig. diapositiva)

Ejemplo de conversión (casting)

```c
#include <stdio.h>

int main () {
    int x = 3;
    int *p1;
    void *p2;
    int *p3;

    p1 = &x;
    printf ("El valor de x es %d \n", *p1);
    p2 = &x;
    //printf ("El valor de x es %d", *p2);

    printf ("El valor de x es %d \n", * (int*) p2);
    //printf ("El valor apuntado por p3 es %d \n", *p3);
    return 0;
}
```

da error de compilación (indirección invalida)

Convierto el puntero a tipo int

Que pasa en este caso?????

Memoria dinámica

- Se puede reservar o liberar durante la ejecución de un programa de acuerdo a las necesidades
- Se diferencia de la que usábamos hasta ahora (estática) cuyo tamaño quedaba fijo al escribir y compilar el código. Por ejemplo:

```
int v [1000];
```

El espacio que se va a reservar queda fijo al escribir el programa

Memoria dinámica

- Para reservar memoria dinámica se deben usar las siguientes funciones que se encuentran en la biblioteca stdlib:
  - malloc
    - Reserva un bloque de memoria
  - calloc
    - Reserva un bloque de memoria y lo inicializa con ceros
  - realloc
    - Cambia el tamaño de un bloque reservado
  - free
    - Libera un bloque de memoria reservado dinámicamente

Encabezados <stdlib.h>

```
void *malloc(size_t tamaño_bloque);
void *calloc(size_t cant_bloques, size_t tamaño_bloque);
void *realloc(void *puntero, size_t nuevo_tamaño_bloque);
void free(void *puntero);
```

free NO devuelve nada!!

- Los tamaños se miden en bytes
- El tipo size_t es un tipo entero específico para manejo de memoria. Si no se trabaja con bloques enormes (gigas) es equivalente a int
- malloc, calloc y realloc devuelven un puntero de tipo void*. Si el valor devuelto es NULL es porque no hay suficiente memoria disponible

ver ejemplo_malloc.c y ejemplo_realloc_struct_archivo.c

Espacios de memoria

- La memoria reservada en forma estática (variables locales) se reserva en el stack (pila)
  - Deja de ser válida al salir de la función donde se reservó
- La memoria reservada en forma dinámica se reserva en el heap (montón)
  - Es válida hasta que se libere explícitamente con la función free()

(ver ejemplo stackVsHeap.c)