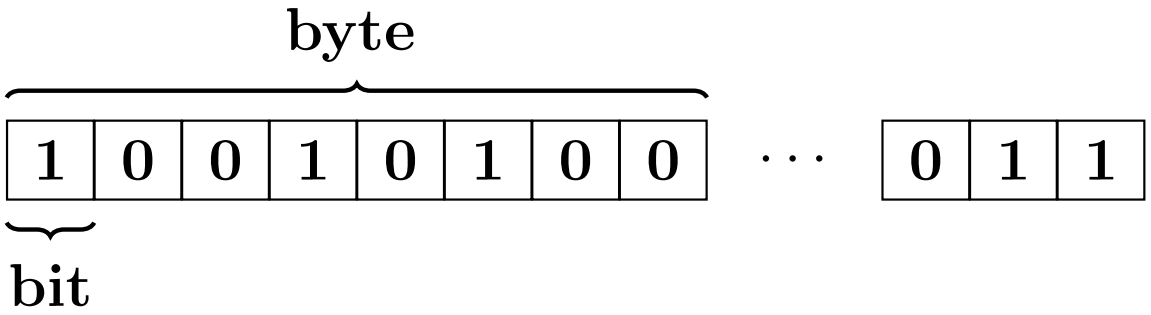
Memoria y Apuntadores

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2019-II

Bits y bytes



- Bit: Contiene un 0 o un 1
- Byte: 8 bits
- Word: Usualmente 4 bytes (tamaño de `int`)

Bases binaria y hexadecimal

Binaria (Bin)

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$$

Bases binaria y hexadecimal

Binaria (Bin)

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$$

Hexadecimal (Hex)

"Dígitos": `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e` *y* `f`

$$\texttt{1c3a} = 1 \times 16^3 + 12 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = 7226$$

Bases binaria y hexadecimal

Binaria (Bin)

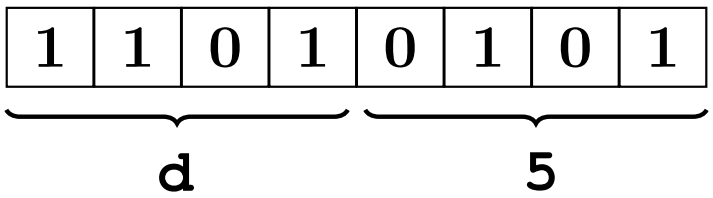
$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$$

Hexadecimal (Hex)

"Dígitos": `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e` *y* `f`

$$\texttt{1c3a} = 1 \times 16^3 + 12 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = 7226$$

Bin a Hex:



Direcciones de memoria se representan en Hex!

Representando otros tipos

- `bool`: `false` (0) o `true` (1)
- `char`: 0 a $2^8 - 1 = 255$ (1 byte)
- `unsigned int`: 0 a $2^{32} - 1 = 4294967295$ (4 bytes)
- `int`: -2^{31} a $2^{31} - 1$
  (4 bytes: 1 bit para el signo y 31 para la mantisa)
- `float`: $-2^{23} \times 10^{2^7-1}$ a $(2^{23} - 1) \times 10^{2^7-1}$
  (4 bytes: 1 bit para el signo, 23 bits para la mantisa, 1 bit para el signo del exponente y 7 bits para el exponente)

Representando otros tipos

Para obtener el tamaño en bytes de un tipo de datos

Tamaño del tipo de dato

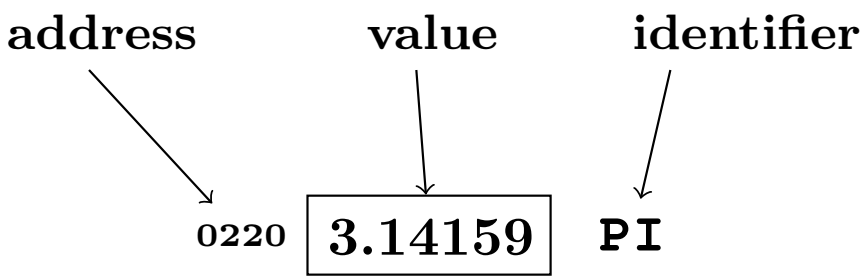
```
sizeof(tipo_de_dato)
```

Direcciones de memoria

- Cada byte en la memoria está identificado por una **dirección**
- Las direcciones van de 0 al No. de bytes en la máquina menos uno
- Las direcciones se dan en formato Hex. Ej. Memoria de 64KB: Desde 0 (**0000**) a $2^{16} - 1 = 65535$ (**FFFF**)

Asignación de memoria

```
const float PI = 3.14159;
```



Asignación de memoria

Static: Administrado estrictamente por el procesador. Tamaño limitado. Acceso rápido. Guarda el código compilado y las variables globales y estáticas

Asignación de memoria

Static: Administrado estrictamente por el procesador. Tamaño limitado. Acceso rápido. Guarda el código compilado y las variables globales y estáticas

Stack: Administrado estrictamente por el procesador. Tamaño limitado. Acceso rápido. Guarda variables locales y las libera cuando salen de cobertura (scope)

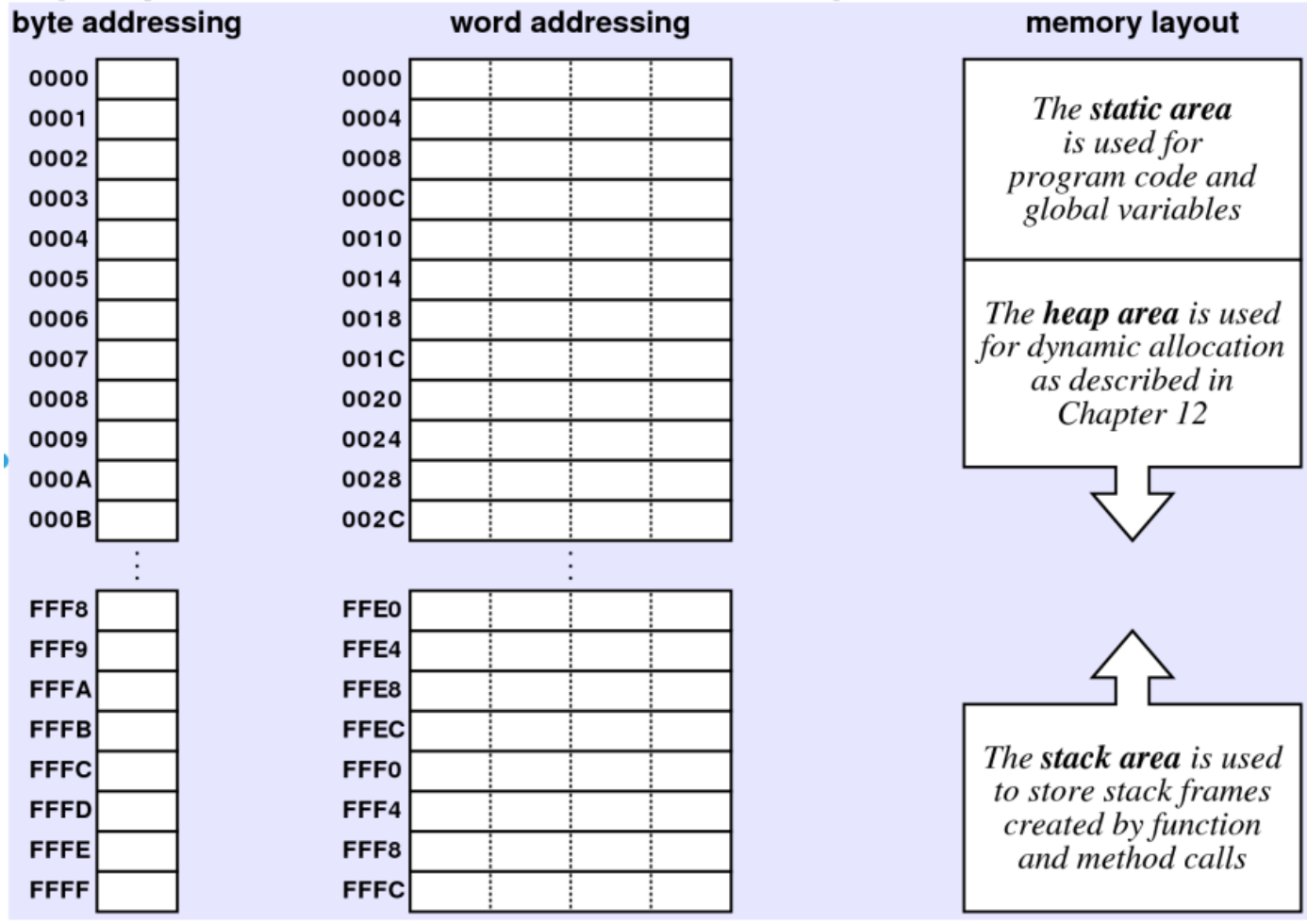
Asignación de memoria

Static: Administrado estrictamente por el procesador. Tamaño limitado. Acceso rápido. Guarda el código compilado y las variables globales y estáticas

Stack: Administrado estrictamente por el procesador. Tamaño limitado. Acceso rápido. Guarda variables locales y las libera cuando salen de cobertura (scope)

Heap: Administrado por el usuario. Tamaño limitado por la memoria física. Cobertura (scope) global

Asignación de memoria



Asignación de memoria

Tomemos el siguiente código:

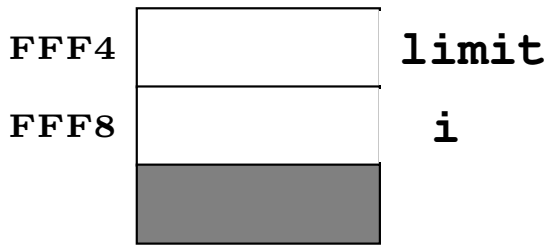
```
int main() {
  int limite;
  cout << "Este programa lista las potencias de dos."
       << endl;
  cout << "Ingrese el exponente limite: ";
  cin >> limite;
  for (int i = 0; i <= limite; i++) {
    cout << "2 a la " << i << " = "
         << elevarAPotencia(2, i) << endl;
  }
  return 0;
}
```

Asignación de memoria

```
int elevarAPotencia(int n, int k){
  int result = 1;
  for (int i = 0; i < k; i++) {
    result *= n;
  }
  return result;
}
```

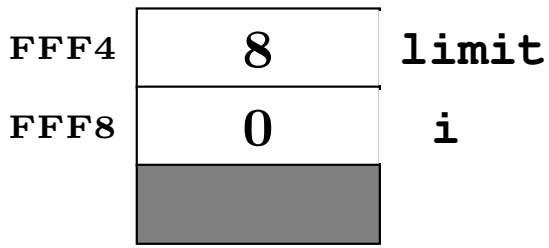
Asignación de memoria

Primero, `main` separa memoria para sus variables locales en el stack:



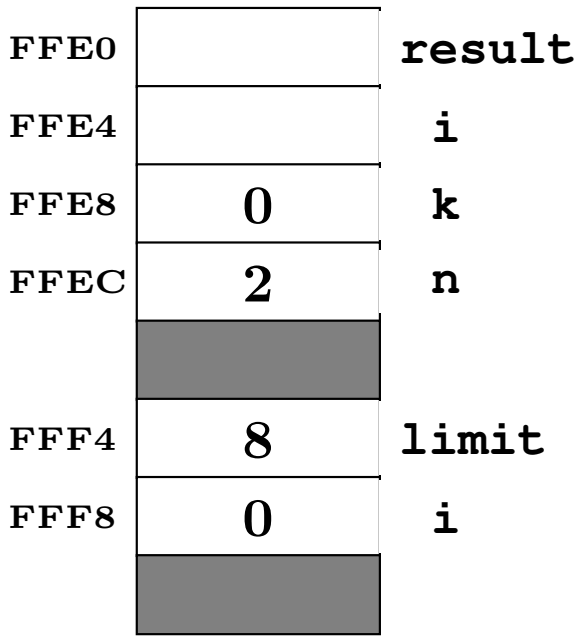
Asignación de memoria

Luego se asignan valores a las variables:



Asignación de memoria

Al llamar `elevarAPotencia` se separa la memoria e inicializan los parámetros:



Asignación de memoria

Cuando `elevarAPotencia` retorna, se eliminan sus variables locales y se libera la memoria del stack.

Apuntadores

xkcd.com/138/

Apuntadores

xkcd.com/138/

- Guardan direcciones de memoria

Apuntadores

xkcd.com/138/

- Guardan direcciones de memoria
- Se refieren a estructuras grandes de forma compacta

Apuntadores

xkcd.com/138/

- Guardan direcciones de memoria
- Se refieren a estructuras grandes de forma compacta
- Usados para reservar memoria durante la ejecución

Apuntadores

xkcd.com/138/

- Guardan direcciones de memoria
- Se refieren a estructuras grandes de forma compacta
- Usados para reservar memoria durante la ejecución
- Guardan relaciones entre los datos

Apuntadores

Declaración

*qualifier type *pointer_name = &name*

Ejemplo

```
int myVariable = 5;
int *myPointer;
myPointer = &myVariable;
```

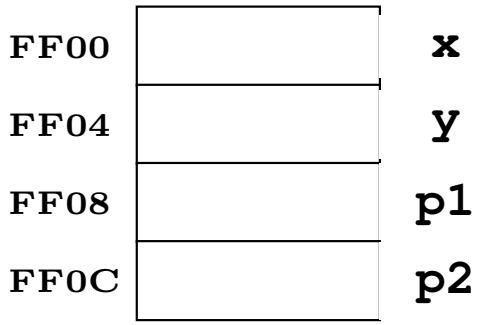
*: Valor al que se apunta

&: Dirección de

Apuntadores

Considere las declaraciones:

```
int x, y;
int *p1, *p2;
```



Apuntadores

```
x = 42;
y = 163;
```

FF00	42	x
FF04	163	y
FF08		p1
FF0C		p2

Apuntadores

```
p1 = &y;
p2 = &x;
```

FF00	42	x
FF04	163	y
FF08	FF04	p1
FF0C	FF00	p2

Apuntadores

El operador $\star$ se usa para acceder al valor al que apunta un apuntador

```
*p1 = 17;
```

FF00	42	x
FF04	17	y
FF08	FF04	p1
FF0C	FF00	p2

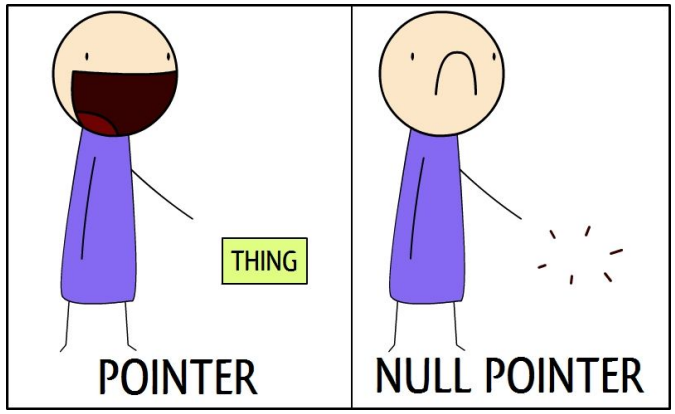
Apuntadores

Se puede copiar un apuntador

```
p1 = p2;
```

FF00	42	x
FF04	17	y
FF08	FF00	p1
FF0C	FF00	p2

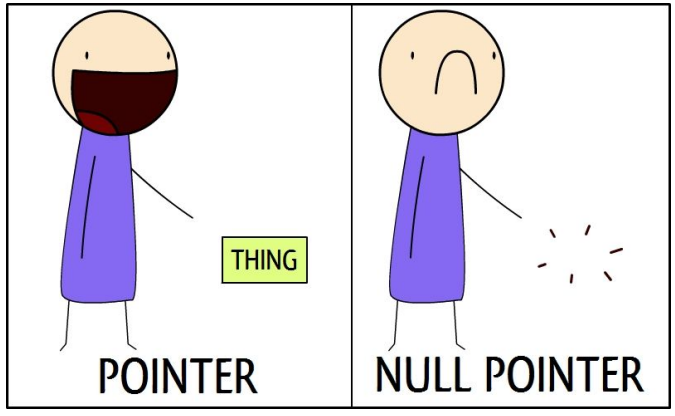
Apuntadores



somethingofthatilk.com

Apuntador `nullptr`:

Apuntadores

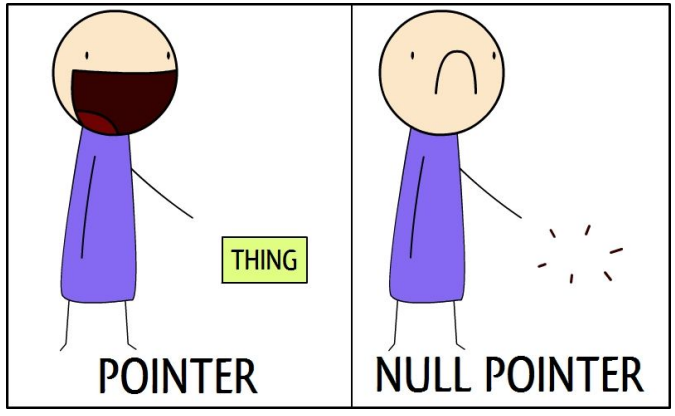


somethingofthatilk.com

Apuntador `nullptr`:

- Indica que el apuntador no se refiere a ninguna dirección válida

Apuntadores



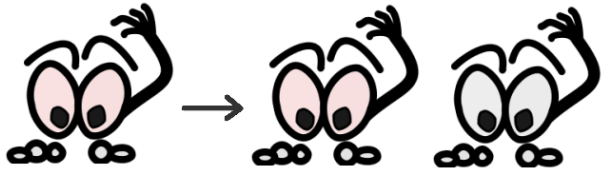
somethingofthatilk.com

Apuntador `nullptr`:

- Indica que el apuntador no se refiere a ninguna dirección válida
- Es ilegal usar $\star$ en un apuntador nulo

Pasar argumentos por valor y por referencia

Paso por referencia



Paso por valor

Image: https://www.elsieisy.com/where-do-you-live/

Pasar argumentos por valor y por referencia

Paso por referencia

```
void foo(int &var){
  var++;
}

int main(){
  int x = 3;
  foo(x);
  return 0;
}
```

Pasar argumentos por valor y por referencia

Paso por valor

```
void foo(int var){
  var++;
}

int main(){
  int x = 3;
  foo(x);
  return 0;
}
```