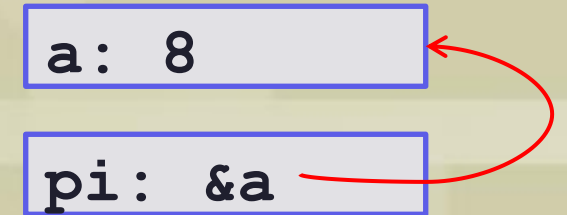


Que vimos? Punteros

Nuevos tipos de variables que sirven para almacenar una dirección de memoria. Los punteros tienen un **tipo**, que es el tipo de dato que se encuentra en la dirección de memoria apuntada.

```
int a = 8;  
int *pi = &a;
```



pi es un puntero a entero, también se puede leer que el contenido de la memoria apuntada por **pi** es un entero.

```
double *pd2 = &a; <<< Error, "a" es un entero, no un double
```

Punteros

```
#include "icom_helpers.h"
```

```
int main()
```

```
{
```

```
    int count = 10, x;
```

```
    int *pi;
```

```
    pi = &count;
```

```
    x = *pi;
```

```
    cout << "count = " << count << " x = " << x << endl;
```

```
    return 0;
```

```
}
```

```
count = 10 x = 10
```

count: 10
x: ???
pi: ???

count: 10
x: ???
pi: &count

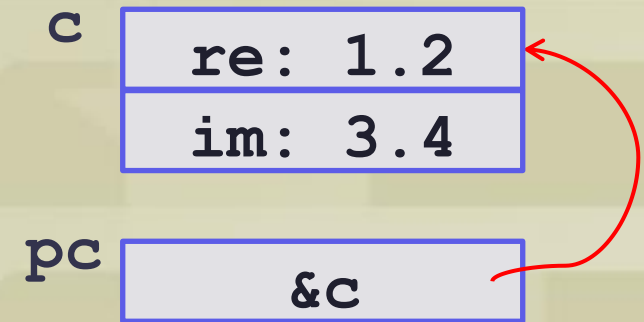
count: 10
x: 10
pi: &count

Punteros a UDT

- Definición similar a un puntero a un tipo nativo:

```
struct Complex {  
    double re;  
    double im;  
};
```

```
Complex c {1.2, 3.4};  
Complex *pc;  
pc = &c;
```



- Nuevo operador de selección de campo: ->

```
pc->re = 8;  
cout << "re: " << pc->re << " im: " << pc->im << endl;
```

- $pc \rightarrow re \equiv (*pc).re$

Punteros y arreglos nativos

- Un arreglo nativo se comporta como es un puntero constante al primer elemento del vector.

```
int vec[100];  
int *pi = vec;    // int *pi = &vec[0];
```

- Los arreglos nativos son punteros que no pueden ser “reapuntados”.

```
int a, v1[10], v2[10], *pi;  
pi = &a;  
pi = v1;  
pi = &v2[3];  
v2 = v1; <<< Error: v2 es como un puntero, pero no lo puedo reasignar
```

- Se puede derreferenciar un puntero como un arreglo nativo.
`a = pi[5];`
- Los arreglos nativos son tales cuando se definen, cuando se pasan como argumento a una función van como un puntero, que no es mas que la copia de la dirección de memoria del primer elemento del vector.

Operaciones con Punteros

- Comparar 2 punteros con los operadores ==, !=, <, >, <= ó >=, el resultado es un valor verdadero o falso.

```
int a, b, *pia = &a, *pib = &b;
if( pia == pib )
    cout << "UPS! problemas..." << endl;
```

- Suma o resta de un entero a un puntero, el resultado es un puntero.

```
int vec[10], *pi = vec, *pi2;
pi2 = pi + 2;
*pi2 = 3;
*(pi + 1) = 7;
```

- Resta de 2 punteros del mismo tipo. El resultado es un entero con signo (entero de tipo ptrdiff_t).

```
int n, vec[10], *pi = &vec[0], *pi2 = &vec[5];
n = pi2 - pi;
```

Mas de punteros y arreglos nativos

- Los punteros se pueden derreferenciar como los arreglos nativos.

```
int a, *ptr, v[] = { 0,1,2,3,4,5 };  
ptr = v;  
a = ptr[2];  
ptr[1] = 567;
```

- Equivalencias.

$$*(ptr + i) \equiv ptr[i]$$
$$(ptr + i) \equiv \&ptr[i]$$

Punteros a función

Los punteros a función son punteros que contienen la dirección de memoria donde se encuentra el código de una función. El tipo de un puntero a función está dado por el prototipo de la función a la que apunta. Esto es, el tipo de retorno y la cantidad y tipo de sus argumentos.

- Declaración de una variable puntero a función:

```
int (*fnPtr) (void) ;
```

Esto declara la variable `fnPtr` como un puntero a una función que retorna un entero y no recibe argumentos.

- Asignación de un puntero a función:

```
fnPtr = rand;
```

- Llamado a una función a través de un puntero a función:

```
int r = fnPtr();          // equiv. A r = rand();
```

Punteros a función como argumento

El tipo de puntero a función lo utiliza el compilador para asegurar que al ser llamada la función, la cantidad y el tipo de los argumentos y el tipo del valor de retorno sean los adecuados.

```
#include "icom_helpers.h"

using fun_ptr_t = double (*)(double);

double integra(double x0, double x1, fun_ptr_t fun);

int main() {
    double i;

    i = integra(0, M_PI, sin);
    i = integra(0, M_PI, cos);

    return 0;
}

// double integra(double x0, double x1, double(*fun)(double));
```