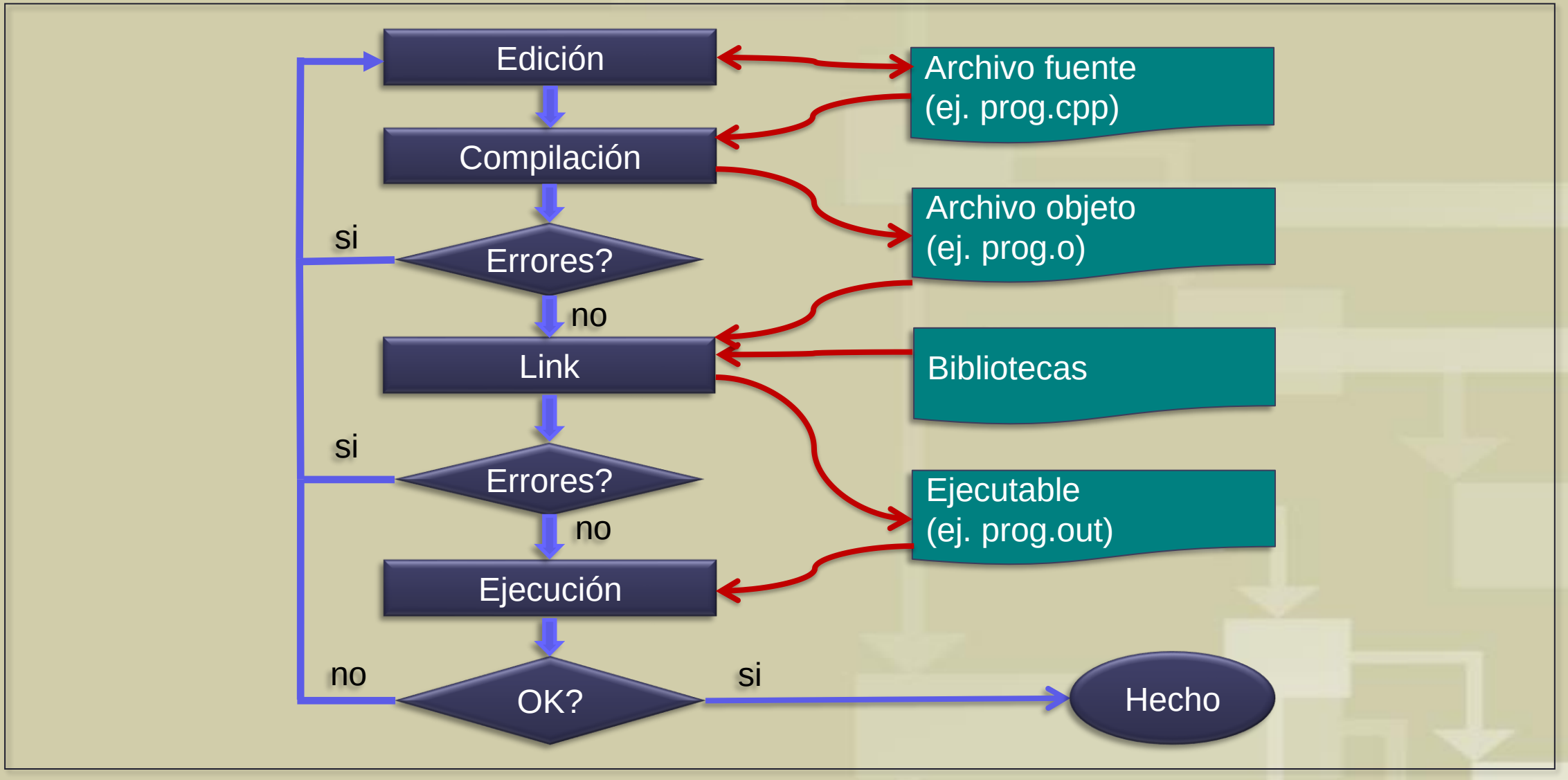
Primeros pasos con el lenguaje

Nivel de abstracción

➢Lenguajes de alto nivel. Permiten la manipulación de las entidades del problema sin involucrarse con los detalles de su implementación interna. (**FORTRAN**, **Pascal**, **C**, **Basic**, **Java**, **C++, Objective C**, **C#,** etc. , etc. , etc.). Portabilidad.

```
Port_t ToMemMgr::portMRAlloc (u8 szLog)
{
    Port_t ret = rdFree[szLog];
    rdFree[szLog] = MRDescTab[ret].refStrAddr;

    return ret;
}
```

Generación de un programa



Estructura de un programa en C++

- Comentarios.
- Directivas al preprocesador.
- Definiciones de tipos de datos.
- Declaraciones de variables.
- Definiciones de funciones.

Ejem3_1.cpp

```
/* Primer programa en C++ */

#include "icom_helpers.h"

int main()    // función que toma control cuando arranca el programa
{
    cout << "hola mundo!\n";  // imprime hola mundo! Seguido de un
                              // delimitador de línea
    return 0;                 // retorna un valor entero indicando
                              // éxito o fallo
}
```

Variables/Objetos

- En lenguajes de bajo nivel de abstracción, la representación de datos y su manipulación involucran la interacción directa con el hardware (sumamente tedioso).

- Los lenguajes de alto nivel brindan una abstracción que facilita estas tareas: **la variable**:

“Relación entre un nombre simbólico, una porción de memoria y un tipo.”

- Las variables tienen asociado un bloque de memoria (dirección de comienzo y tamaño), además tienen un tipo que define la representación y las operaciones que pueden realizarse sobre ella. En el contexto de POO se dice que una variable es un objeto o instancia de una clase.

Operaciones entre enteros y strings

Strings	Enteros y punto flotante
`cin >> w` : lee una palabra y la pone en `w`	`cin >> v` : lee un valor numérico y lo pone en `v`
`cout << w` : escribe la palabra almacenada en `w`	`cout << v` : escribe el valor almacenada en v
`w+s` : concatena de `w` y `s`	`v+x` : suma `v` y `x`
`w+=s` : agrega `s` al final de `w`	`v+=x` : incrementa el valor de `v` en `x`
…	`v++` : incrementa `v` en `1`
	`v-x` : resta
	…

El tipo de una variable determina cuales operaciones son válidas y su significado para ese tipo (esto se denomina sobrecarga y sobrecarga de operadores)

Nombre de variables

- Los nombres de variables en C++ deben comenzar con una letra o el carácter '_' seguida por cualquier combinación de letras (mayúsculas o minúsculas), '_' o dígitos (0-9).

Nombres válidos	Nombres inválidos
`sumValue`	`sum$value`
`is_even`	`is even`
`_sysFlag`	`3Flag`
`J5x7`	`int`

- Se recomienda no utilizar nombres que comiencen con '_', ya que son reservadas para implementaciones y entidades de sistema.

- No se pueden utilizar palabras reservadas (**`int`**, **`if`**, **`while`**, **`for`**, **`double`**, etc.)

Tipos y constantes literales

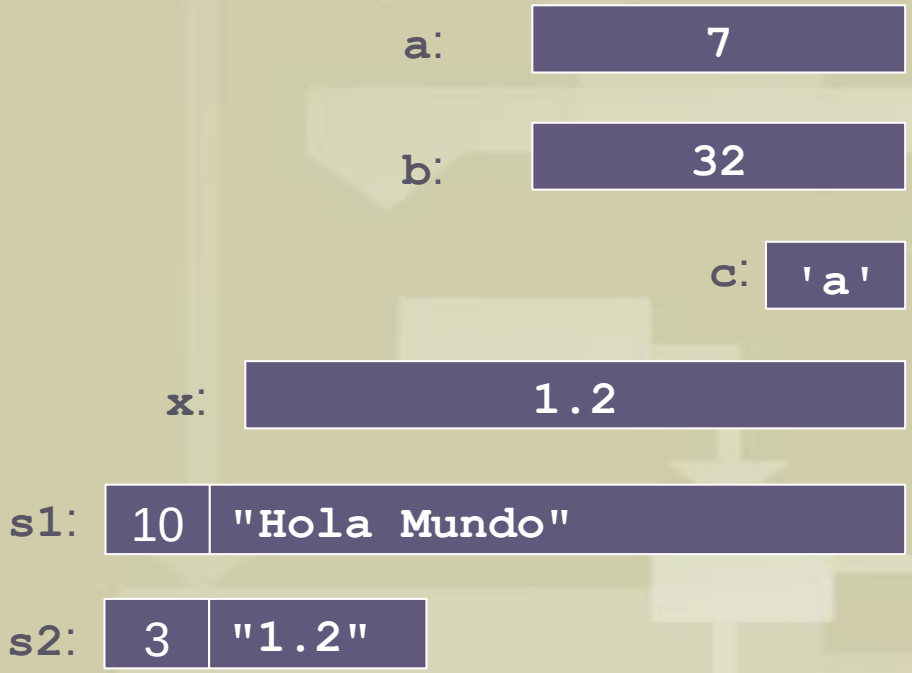
Tipos Nativos	literales
Booleanos: `bool`	`true`, `false`
Caracteres: `char`	`'a'`, `'X'`, `'4'`, `'\n'`, `'$'`
Enteros: `int` (y `short` y `long`)	`0`, `1`, `456`, `-7`, `034`, `0xA4`
Punto flotante: `double` (y `float`)	`1.2`, `3.141592`, `.5`, `-0.54`, `1.2e3`, `.3F`

Tipos de bibliotecas estandar	literales
`string`	`"este es un string literal"`
`vector<int>`	`{1,2,3}`
…	…
…	…

Declaración e inicialización

- Toda variable tiene asociada una porción de memoria y un tipo.
- El tipo define la representación y las operaciones que se pueden llevar a cabo
- A una variable de un tipo se la denomina **instancia de tipo** u **objeto**

```
int a = 7;

int b = 32;

char c = 'a';

double x = 1.2;

string s1 = "Hola Mundo";

string s2 = "1.2";
```



Operadores

- Algebráicos

  + - * / % =

  += -= *= /= %=

  ++ --

- Relacionales

  > < >= <= == !=

- Lógicos

  && || !

- De bits

  & | ^ >> << ~

  &= |= ^= >>= <<=

- De referenciamiento

  & *

Tipos definidos por el usuario (UDT)

- C++ permite que el usuario (programador) defina sus propios tipos. Ejemplo holístico:

Ejem3_11.cpp

```
#include "icom_helpers.h"
struct Complejo {
    double re, im;
    void print() {
        cout << "(" <<re << " + i " << im << ")\n";
    }
    double modulo() {
        return sqrt(re*re+im*im);
    }
    Complejo operator+(Complejo a) {
        Complejo result = { re + a.re, im + a.im};
        return result;
    }
};

int main() {
    Complejo c1 = {1.2, 3.4}, c2 = {5.6, 7.8},
             c3 = c1 + c2;
    c3.print();
    return 0;
}
```