



# Modularidad

Universidad Católica Boliviana

MSc, José Jesús Cabrera Pantoja

# Outline

- Modularidad

# Separar el codigo

Verificar el password



Manejar errores



Calcular el total a pagar



Mostrar el menú de un  
restaurante



# Funciones

- Agrupar el código según un criterio
- Separar el código por propósito
- Reutilizar el código en varias partes del programa

# Creando funciones

Nos ayuda a

- Organizar
- Modificar
- Entender
- NO reinventar la rueda
- **DONT REPEAT YOUR SELF**

Nombre de nuestra funcion



# Creando funciones

## Otros nombres

- Modulos
- Metodos

Nombre de nuestra funcion



# Creando funciones



```
1 # Mas codigo
2 ...
3 instruccion 1
4 instruccion 2
5 instruccion 3
6 instruccion 4
7 instruccion 5
8 instruccion 6
9 ...
10 # Mas codigo
```

# Prototipo de la función: Ejemplo



```
1 int add_numbers(int a, int b);
```



```
1 string some_function(int a, string b);
```



# Prototipo de la función: Void

- La función puede no devolver ningún resultado. Para ello se utiliza el tipo de dato void.
- Esta función no recibe ningún parámetro y no devuelve un resultado.



```
1 void other_function();
```

# Cuerpo de la funcion

- El cuerpo de la función es la implementación. Es decir, la lista de instrucciones que debe ejecutar la función cada vez que esta es llamada.



```
1 int add_number(int a, int b) {  
2     int sum = a + b;  
3     cout << "Hello from here" << endl;  
4     cout << "Another line of code" << endl;  
5     // return the sum  
6     return sum;  
7 }
```

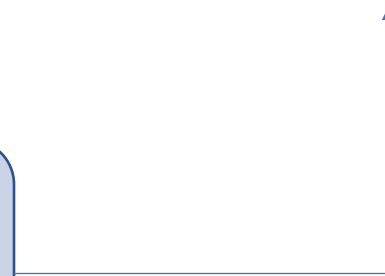
# Cuerpo de la función

- El cuerpo de la función es la implementación. Es decir, la lista de instrucciones que debe ejecutar la función cada vez que esta es llamada.



```
1 int add_number(int a, int b) {  
2     int sum = a + b;  
3     cout << "Hello from here" << endl;  
4     cout << "Another line of code" << endl;  
5     // return the sum  
6     return sum;  
7 }
```

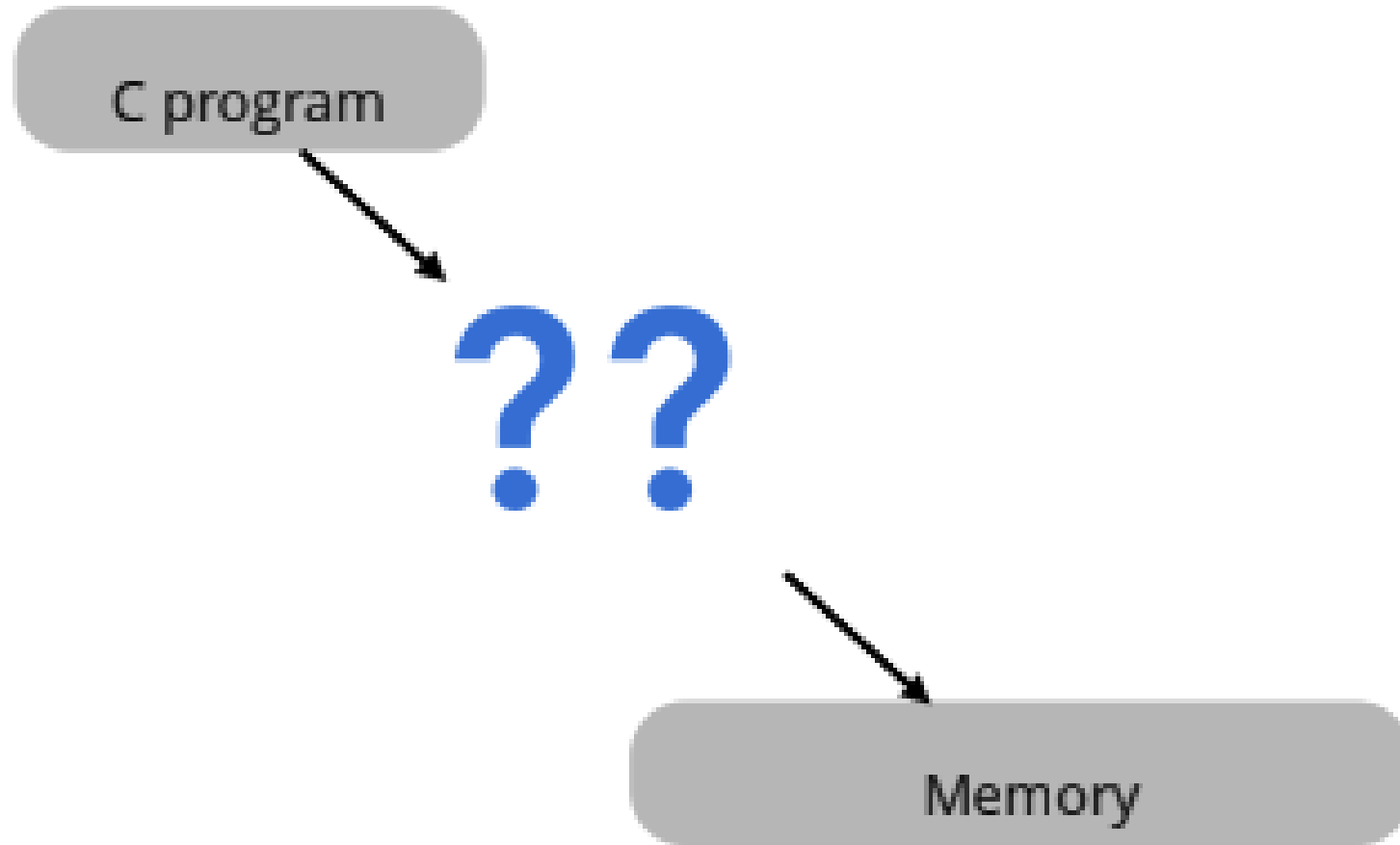
Cuerpo de la función. Instrucciones a realizar



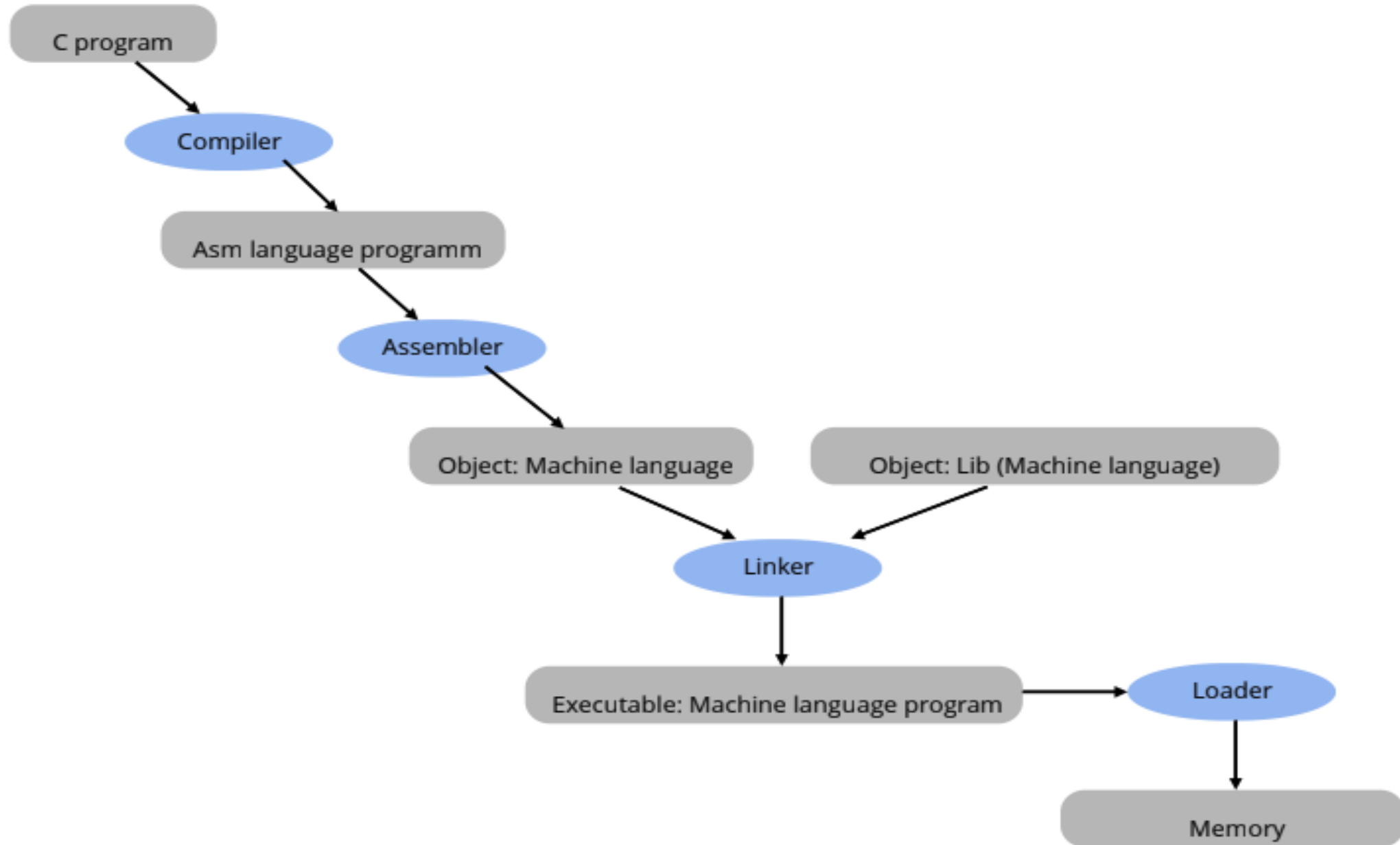
- Esto se le llama declaración de una función.

# Proceso de compilacion

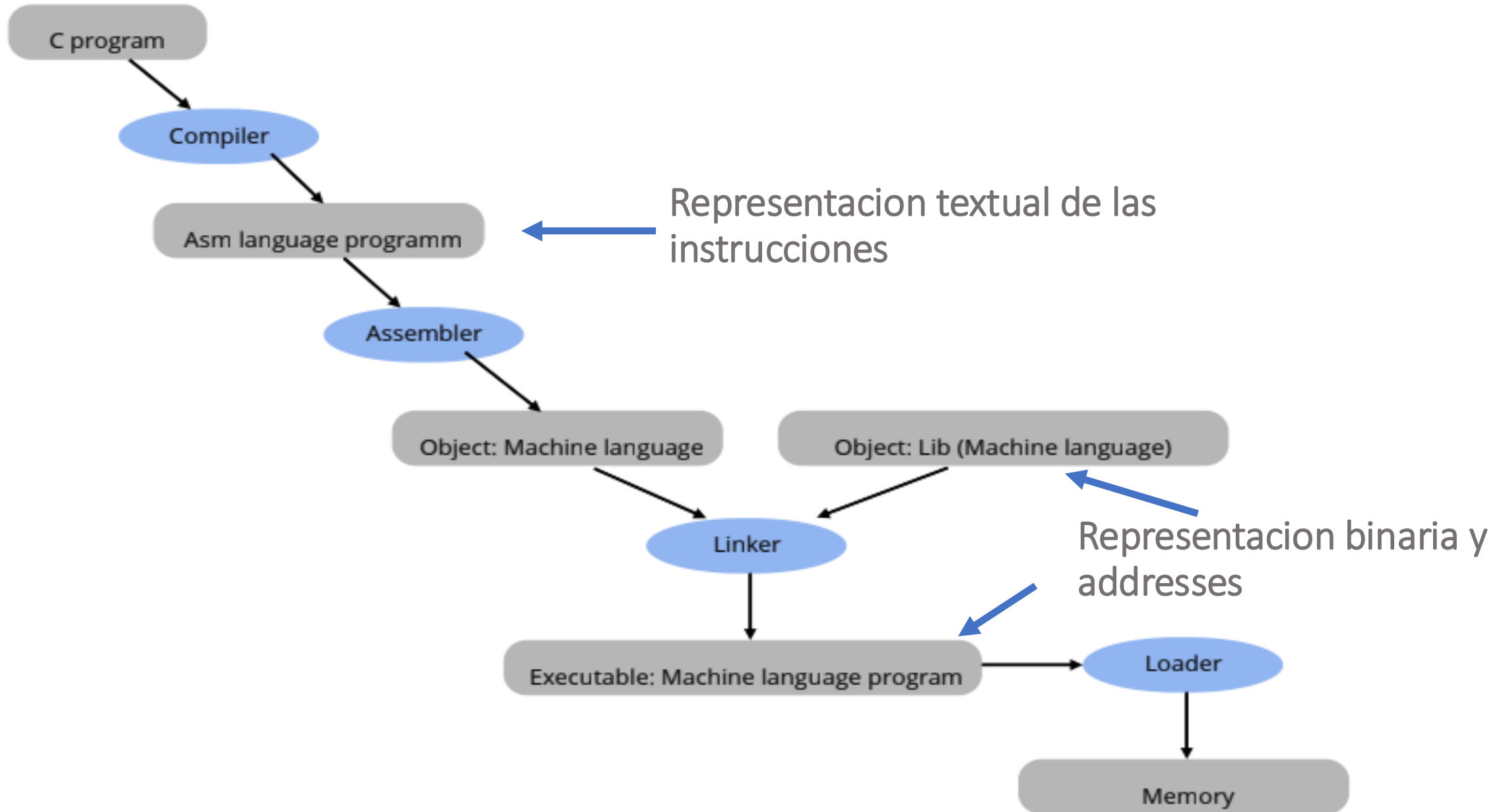
# Proceso de compilacion



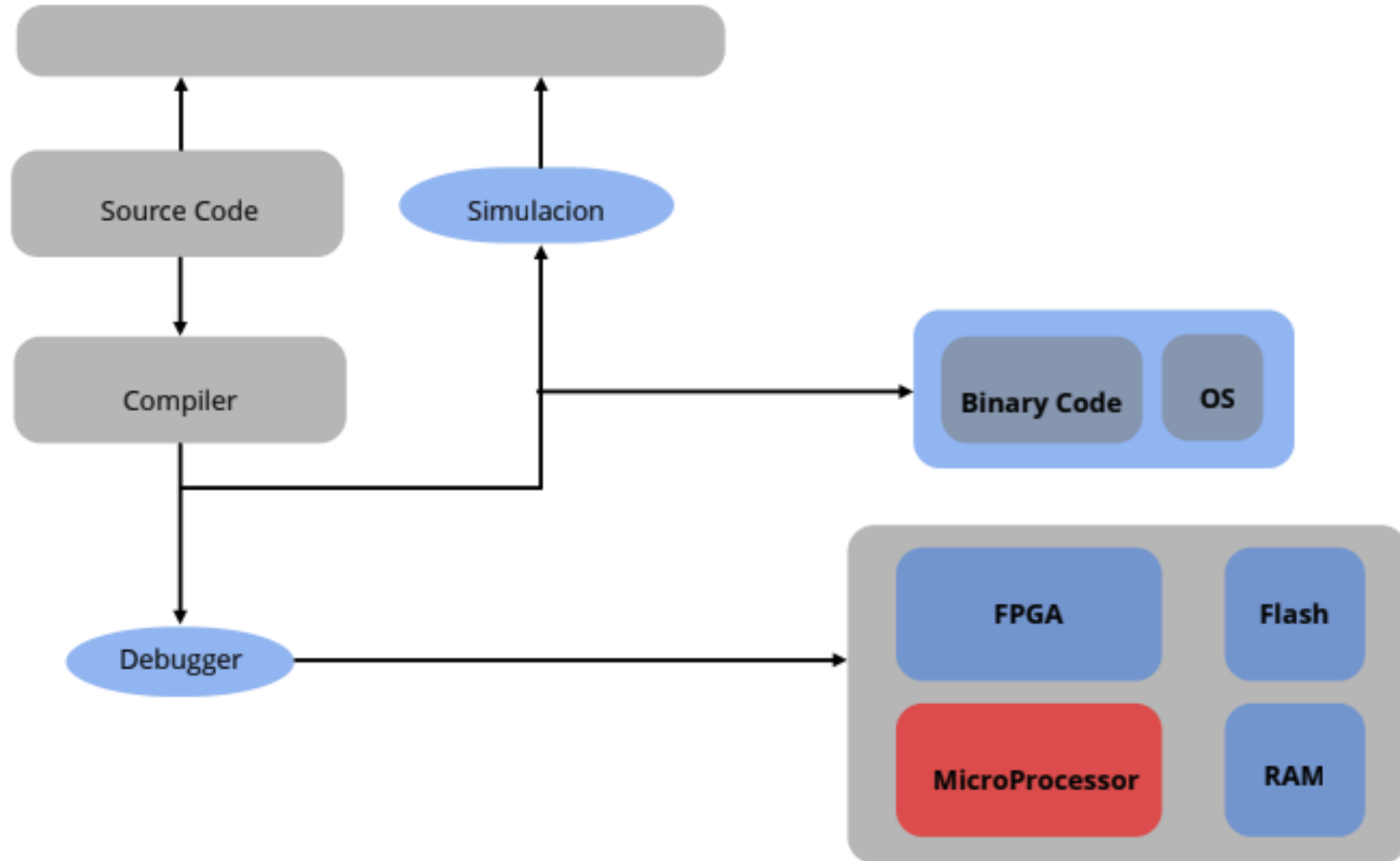
# Proceso de compilacion



# Proceso de compilacion

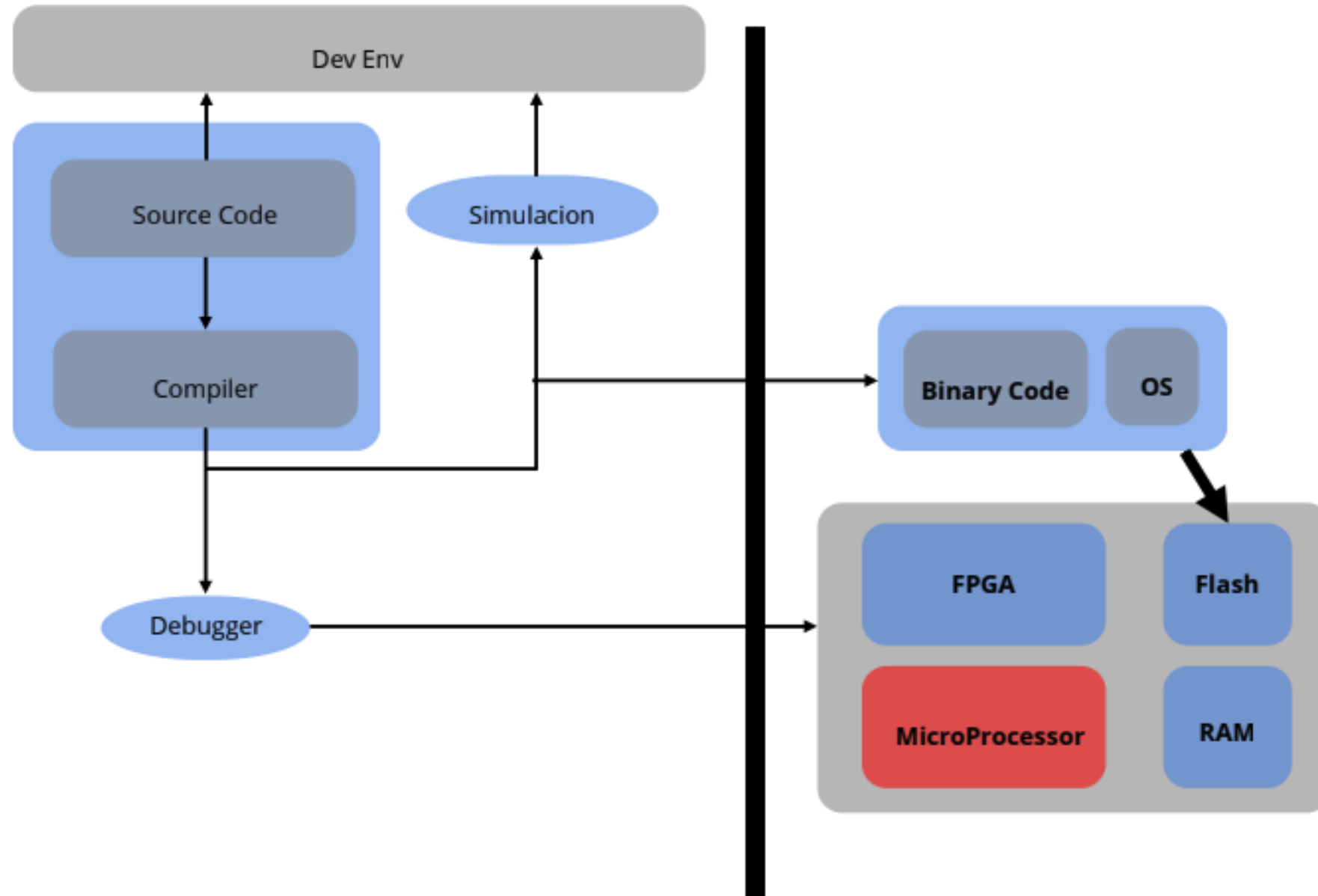


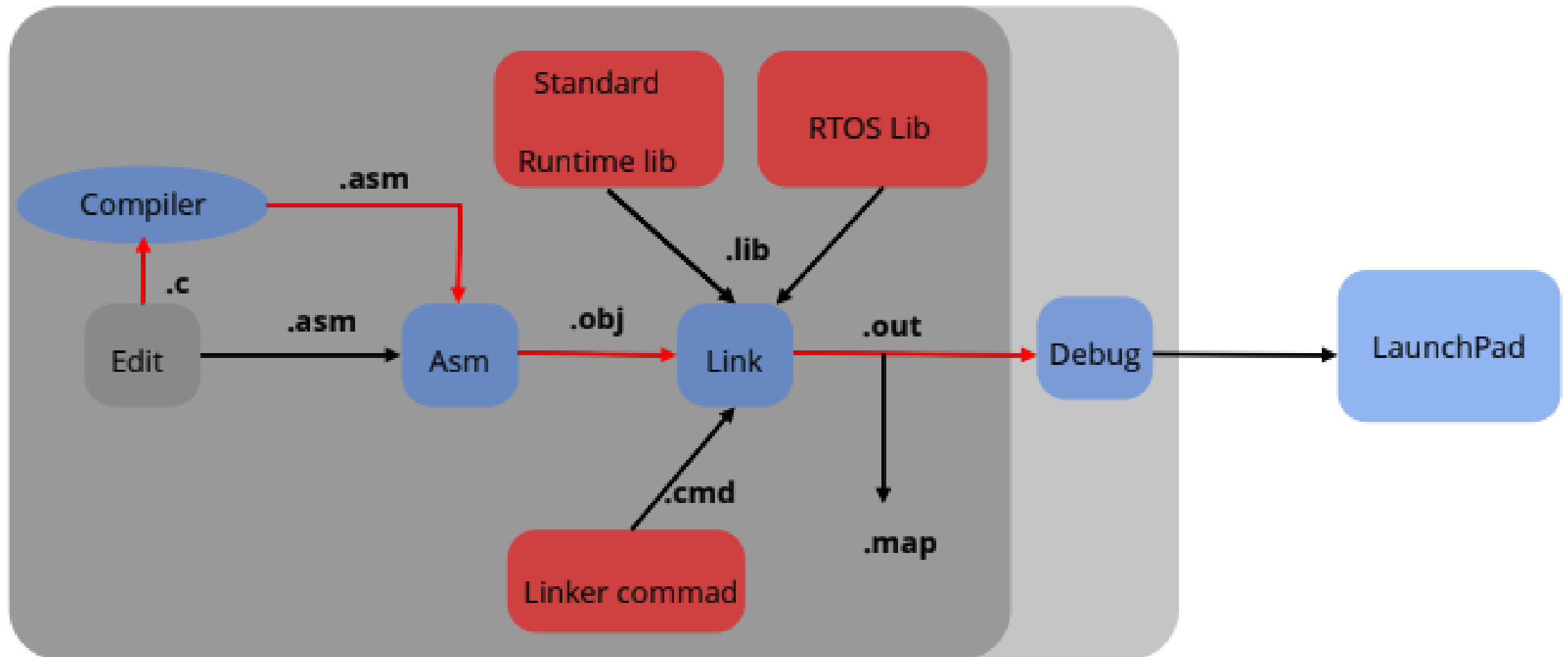
# Proceso de compilacion





# Proceso de compilacion

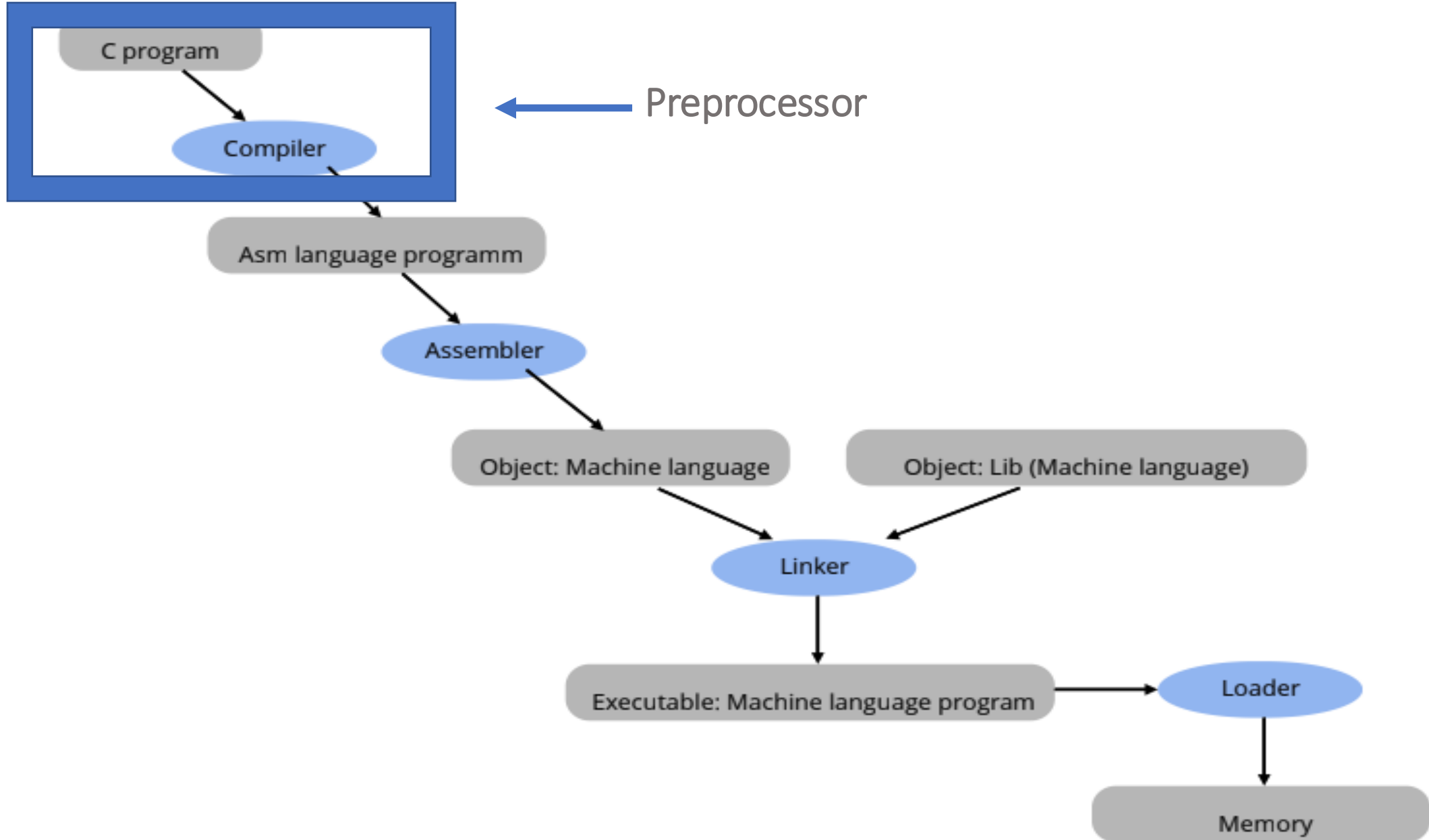




# Assembler

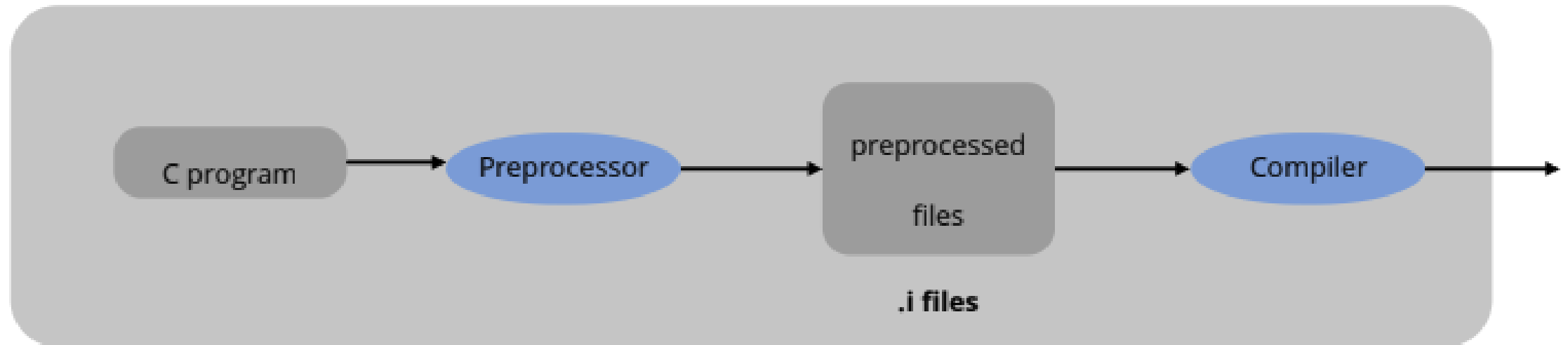
```
...  
1  MOV      R4 , A  
2  MOV      A , R6  
3  SUBB     A , R5  
4  MOV      A , R4  
5  SUBB     A , R0
```

# Proceso de compilacion



# Preprocessor

## Primera etapa



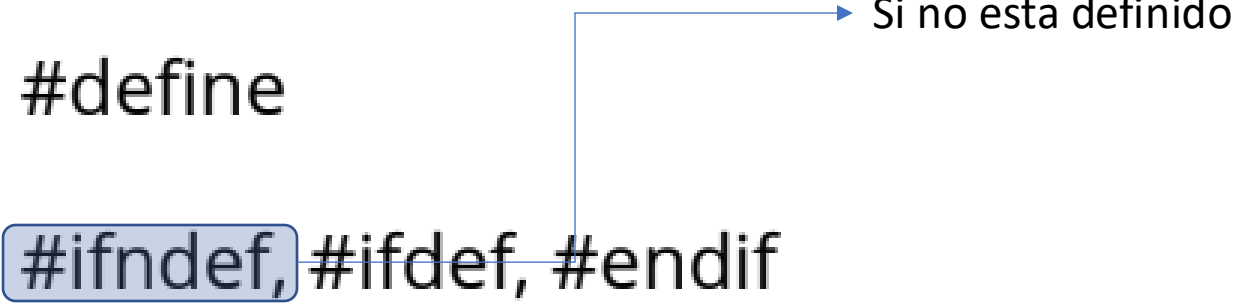
# Preprocessor

Keywords usadas por el preprocesador comienzan con #

- #define
- #ifndef, #ifdef, #endif
- #include
- #pragma
- #error, #warning

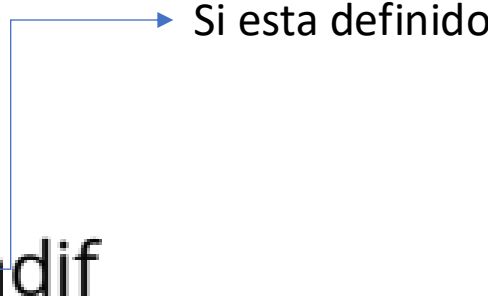
# Preprocessor

Keywords usadas por el preprocesador comienzan con #

- #define
  - #ifndef, #ifdef, #endif
  - #include
  - #pragma
  - #error, #warning
- 
- Si no esta definido

# Preprocessor

Keywords usadas por el preprocesador comienzan con #

- #define
  - #ifndef, #ifdef, #endif
  - #include
  - #pragma
  - #error, #warning
- 
- Si esta definido



# Preprocessor

Keywords usadas por el preprocesador comienzan con #


- #define
- #ifndef, #ifdef, #endif
- #include
- #pragma
- #error, #warning

Fin del if



# Preprocessor

Keywords usadas por el preprocesador comienzan con #

- #define
  - #ifndef, #ifdef, #endif
  - #include
  - #pragma
  - #error, #warning
- 
- Incluir los header (librerías)

# Headers files

- Los headers files son documentos que terminan en .h
- Contienen el prototipo de la función y variables globales que son usadas en el programa
- Es como una “Table de contenido” para nuestro programa, la cual brinda información sobre las funciones que tenemos para usar en el programa

# Headers files


- Los headers files son documentos que terminan en .h
- Contienen el prototipo de la función y variables globales que son usadas en el programa
- Es como una “Table de contenido” para nuestro programa, la cual brinda información sobre las funciones que tenemos para usar en el programa

exercise1.h

```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

# Headers files

Si el header EXERCISE\_H no esta definido entonces se lo define (se lo crea)

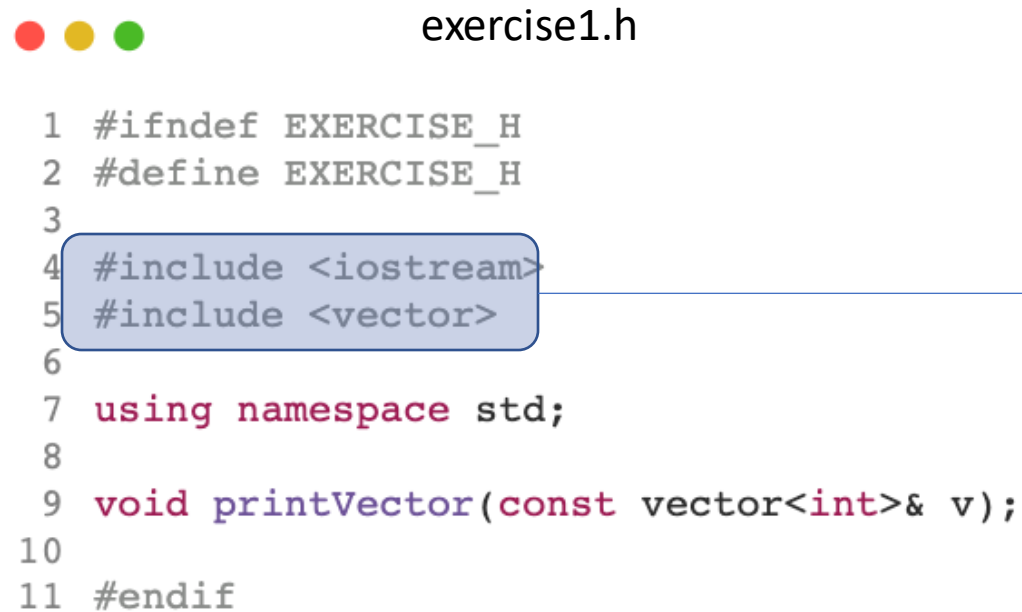


```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

The image shows a code editor window with the title 'exercise1.h'. The code inside the editor is a C++ header file. Lines 1 and 2 are enclosed in a blue rounded rectangle. A blue arrow points from this rectangle to the explanatory text on the right. The code includes standard C++ preprocessor directives for header guards, standard library includes, a namespace declaration, and a function declaration.

# Headers files

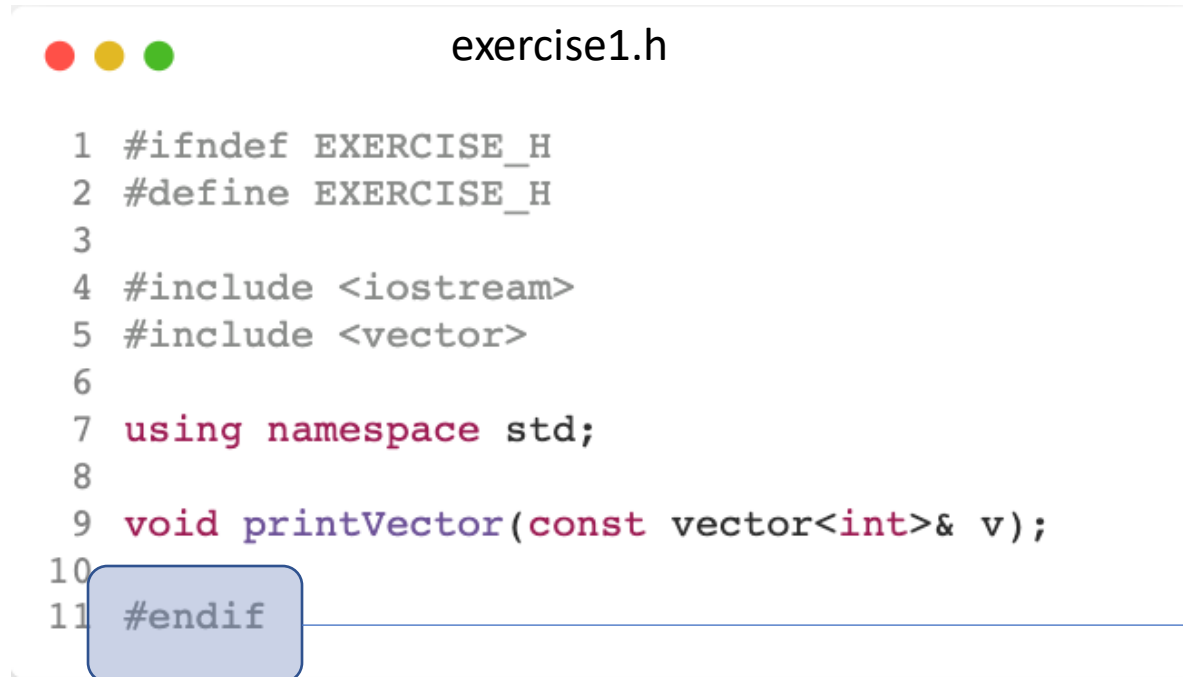
Incluimos las librerías necesarias



```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

# Headers files

Fin del if y fin de la definición del header



```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

# Source files

- Son files con terminación .cpp y crean los ejecutables (.exe)
- En estos files se escribe la implementación de la función
- Es decir tiene el código que describe que hará la función



# Source files

- Son files con terminación cpp
- En estos files se escribe la implementación de la función
- Es decir tiene el código que describe que hará la función

exercise1.cpp

```
1 #include "exercisel.h"
2
3 void printVector(const vector<int>& v) {
4     for (auto e : v) {
5         cout << e << endl;
6     }
7 }
```

Se debe incluir el header file donde esta el prototipo de esta funcion

# Headers y Source files

- Trabajan en conjunto
- Source file – tienen la implementación de las funciones
- Header file – tienen los prototipos de las funciones
- Ejemplo:

# Headers y Source files

- Ejemplo: (Antes de preprocesador)

exercise1.h

```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

exercise1.cpp

```
1 #include "exercise1.h"
2
3 void printVector(const vector<int>& v) {
4     for (auto e : v) {
5         cout << e << endl;
6     }
7 }
```

# Headers y Source files

- Despues del procesador



```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 void printVector(const vector<int>& v) {
12     for (auto e : v) {
13         cout << e << endl;
14     }
15 }
16
17
18 #endif
```

# Headers y Source files

Incluimos las librerías (exercise1.h) en nuestro main. Esto para poder usar las funciones en esta tabla de contenidos.

## exercise1.h

```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

## exercise1.cpp

```
1 #include "exercise1.h"
2
3 void printVector(const vector<int>& v) {
4     for (auto e : v) {
5         cout << e << endl;
6     }
7 }
```

## main.cpp

```
1 #include "exercise1.h"
2
3 int main() {
4     cout << "Hello world" << endl;
5
6     vector<int> va = {1, 2, 3};
7     printVector(va);
8
9     return 0;
10 }
```

# Headers y Source files

Usamos las funciones dentro de exercise1.h

exercise1.h

```
1 #ifndef EXERCISE_H
2 #define EXERCISE_H
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void printVector(const vector<int>& v);
10
11 #endif
```

exercise1.cpp

```
1 #include "exercise1.h"
2
3 void printVector(const vector<int>& v) {
4     for (auto e : v) {
5         cout << e << endl;
6     }
7 }
```

main.cpp

```
1 #include "exercise1.h"
2
3 int main() {
4     cout << "Hello world" << endl;
5
6     vector<int> va = {1, 2, 3};
7     printVector(va);
8
9     return 0;
10 }
```

# Modularidad

Nos ayuda a

- Organizar
- Modificar
- Entender
- NO reinventar la rueda
- **DONT REPEAT YOUR SELF**

headers y source files



# Separar el código

Agrupamos por propósito. El conjunto de funciones que realizan la lectura de la base de datos en un header y source files.

Ejemplo: **database.h** y **database.cpp**



Agrupamos otro conjunto de funciones que realizan algunos cálculos sobre vectores y matrices.

Ejemplo: **matrix\_operation.h** y **matrix\_operation.cpp**