

# Práctica de Organización del Computador II

ABI

---

Primer Cuatrimestre 2023

Organización del Computador II  
DC - UBA

## Contratos de función

---

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos**.

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos**.

Es decir, tenemos ciertas garantías sobre cómo se ubican los datos en memoria (*endianness*, *layouts* de structs, arrays, etc)

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos**.

Es decir, tenemos ciertas garantías sobre cómo se ubican los datos en memoria (*endianness*, *layouts* de structs, arrays, etc)

Veamos ahora la forma en la que definimos y nos adherimos a un **contrato de función**.

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada product.

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.



¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.
- Para llamar a `product` hay que proporcionar dos valores: uno de tipo `int32_t*` y otro de tipo `uint32_t`.

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).
- Todo uso o implementación de la función **deben coincidir en tipo devuelto, cantidad y tipo de parámetros**.

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).
- Todo uso o implementación de la función **deben coincidir en tipo devuelto, cantidad y tipo de parámetros**.
- ¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

**Respuesta:** Vamos a tener que definir el alcance de nuestro contrato en términos de la plataforma particular.

¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

**Respuesta:** Vamos a tener que definir el alcance de nuestro contrato en términos de la plataforma particular.

**Corolario:** Los contratos de función en un lenguaje de alto nivel se pueden definir independientemente de la plataforma.

¿Entonces cualquier lenguaje que genere **código objeto y respete el contrato de función** puede interactuar con funciones ubicadas en bibliotecas binarias (código objeto) que adhieran al contrato?



¿Entonces cualquier lenguaje que genere **código objeto y respete el contrato de función** puede interactuar con funciones ubicadas en bibliotecas binarias (código objeto) que adhieran al contrato?

**Respuesta:** Si, para eso vamos a tener que hablar de la **ABI (Application Binary Interface)**

# **Interfaz binaria de aplicación (ABI)**

---

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.

Estos contratos específicos se llamarán **Interfaces Binarias de Aplicación (ABIs)** y definen la forma en que las funciones serán llamadas, cómo se pasan los parámetros y que invariantes estructurales deben hacerse valer.

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.

Estos contratos específicos se llamarán **Interfaces Binarias de Aplicación (ABIs)** y definen la forma en que las funciones serán llamadas, cómo se pasan los parámetros y que invariantes estructurales deben hacerse valer.

La ABI no sólo depende del procesador, el sistema operativo también juega un rol.

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas



Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de `usuarix` (**Convención de llamada**)

**Pregunta:** ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?



**Pregunta:** ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?

**Pregunta:** ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

**Pregunta:** ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

**Respuesta:** Sí (a todo)

**Pregunta:** ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

**Pregunta:** ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

**Respuesta:** Vamos a dividir los registros entre volátiles y no volátiles

**Pregunta:** ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

**Respuesta:** Vamos a dividir los registros entre volátiles y no volátiles

- **Volátiles:** La función llamada no tiene obligación de conservarlos

**Pregunta:** ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

**Respuesta:** Vamos a dividir los registros entre volátiles y no volátiles

- **Volátiles:** La función llamada no tiene obligación de conservarlos
- **No volátiles:** Si la función llamada los cambia debe restaurarlos antes del `return`

La **ABI** que utilizaremos define dos convenciones de llamada:



La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila.**

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila.**
- Otro para 32 bits, **que sólo utiliza la pila.**

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila.**
- Otro para 32 bits, **que sólo utiliza la pila.**

Las convenciones dependen de la arquitectura del procesador y del sistema operativo:

- En x86-64/Linux (64bits) se denomina System V AMD64 ABI
- En x86/Linux (32bits) se conoce como System V i386 ABI

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila.**
- Otro para 32 bits, **que sólo utiliza la pila.**

Las convenciones dependen de la arquitectura del procesador y del sistema operativo:

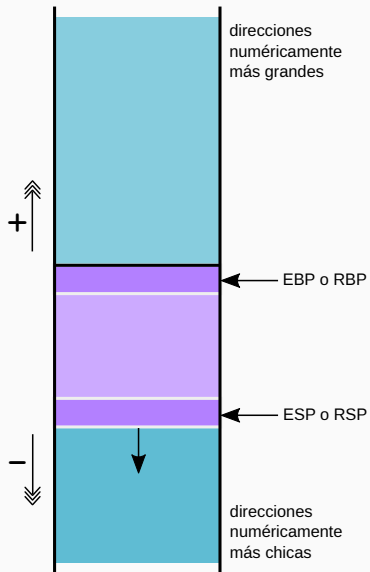
- En x86-64/Linux (64bits) se denomina System V AMD64 ABI
- En x86/Linux (32bits) se conoce como System V i386 ABI

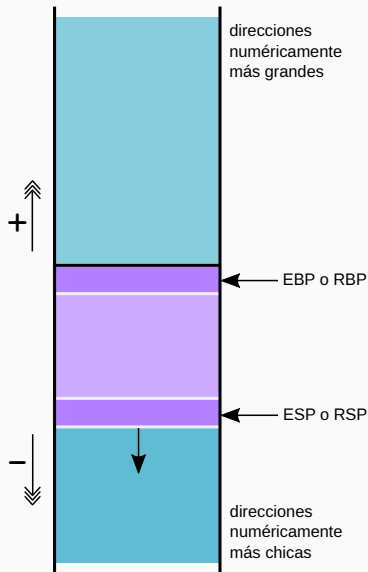
Vamos a usar el primero ahora haciendo programación de aplicaciones y el segundo cuando hagamos programación de sistemas.

## Uso de la pila

---

- La pila es una estructura en memoria
- Sirve para guardar información **local** a una función
- Contiene información de **contexto**: parámetros, dirección de retorno

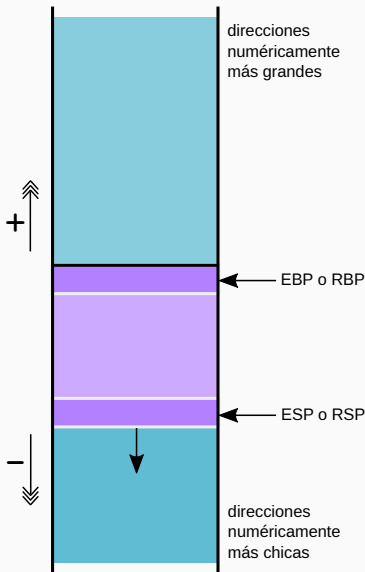




## En 32 bits

- Los registros EBP y ESP
- **EBP (Base Pointer)** apunta a la base
- **ESP (Stack Pointer)** al tope (último elemento válido)



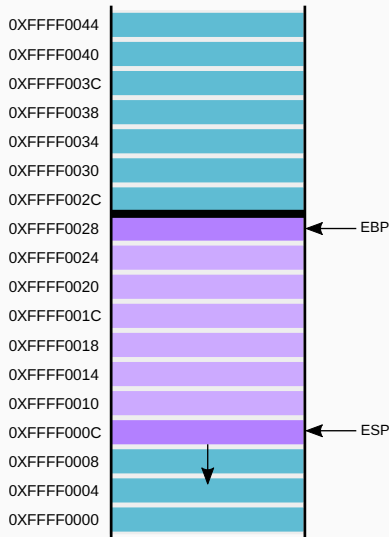


## En 32 bits

- Los registros EBP y ESP
- **EBP (Base Pointer)** apunta a la base
- **ESP (Stack Pointer)** al tope (último elemento válido)

## En 64 bits

- Los registros RBP y RSP
- **RBP (Base Pointer)** apunta a la base
- **RSP (Stack Pointer)** al tope (último elemento válido)



## Alineación

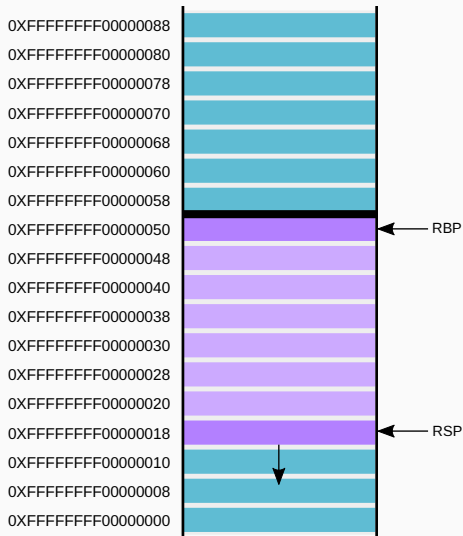
- 4 bytes (en 32bits)
- 16 bytes para llamar funciones de C

## Registros

- ESP
- EBP

## Instrucciones

- PUSH
- POP



## Alineación

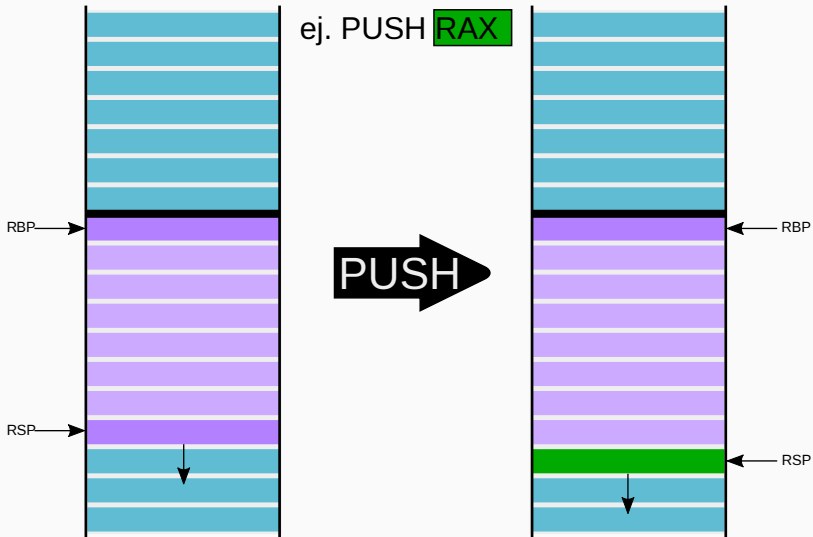
- 8 bytes (en 64 bits)
- 16 bytes para llamar funciones de C

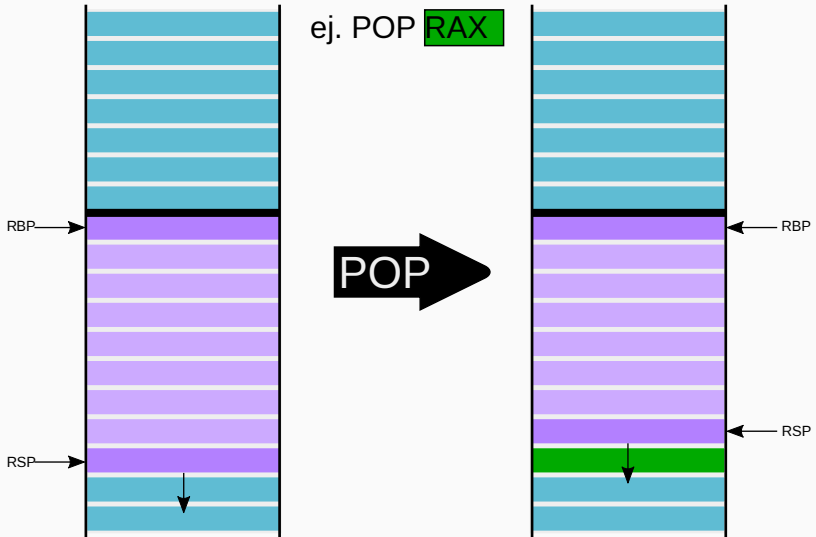
## Registros

- RSP
- RBP

## Instrucciones

- PUSH
- POP





En 64 bits System V define lo siguiente:

---

<sup>1</sup>Para el ABI los punteros son enteros `__int64_t`

En 64 bits System V define lo siguiente:

- Los registros RBX, RBP, R12, R13, R14 y R15 son **no volátiles**

---

<sup>1</sup>Para el ABI los punteros son enteros

En 64 bits System V define lo siguiente:

- Los registros RBX, RBP, R12, R13, R14 y R15 son **no volátiles**
- El valor de retorno será almacenado en RAX para valores enteros<sup>1</sup> y en XMM0 para flotantes

---

<sup>1</sup>Para el ABI los punteros son enteros



En 64 bits System V define lo siguiente:

- Los registros RBX, RBP, R12, R13, R14 y R15 son **no volátiles**
- El valor de retorno será almacenado en RAX para valores enteros<sup>1</sup> y en XMM0 para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP)(¿RSP volátil?)

---

<sup>1</sup>Para el ABI los punteros son enteros

En 64 bits System V define lo siguiente:

- Los registros RBX, RBP, R12, R13, R14 y R15 son **no volátiles**
- El valor de retorno será almacenado en RAX para valores enteros<sup>1</sup> y en XMM0 para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

---

<sup>1</sup>Para el ABI los punteros son enteros

**Aclaración:** Donde diga **de derecha a izquierda** o **de izquierda a derecha** debemos entender que nos referimos al orden de los parámetros en la declaración de la función en el encabezado `.h`.

En 64 bits la System V define lo siguiente:

En 64 bits la System V define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en RDI, RSI, RDX, RCX, R8, R9 respectivamente

En 64 bits la System V define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en RDI, RSI, RDX, RCX, R8, R9 respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7 respectivamente

En 64 bits la System V define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en RDI, RSI, RDX, RCX, R8, R9 respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7 respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH.

En 64 bits la System V define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en RDI, RSI, RDX, RCX, R8, R9 respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7 respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.



En 64 bits la System V define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en RDI, RSI, RDX, RCX, R8, R9 respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7 respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

Para que tengan a mano en la **primera parte de la materia (64 bits)**:

<b>No volátiles</b>	RBX, RBP, R12, R13, R14 y R15 (¿RSP?)
<b>Valor de retorno</b>	RA <sub>X</sub> enteros/punteros, XMM0 flotantes
<b>Entero, puntero</b>	RDI, RSI, RDX, RCX, R8, R9 (izq. a der.)
<b>Flotantes</b>	XMM0, XMM1, . . . , XMM7 (izq. a der.)
<b>¿No hay registros?</b>	PUSH a la pila (der. a izq.)
<b>Inv. de pila</b>	Todo PUSH/SUB debe tener su POP/ADD
<b>Llamada a func.</b>	Pila alineada a 16 bytes

En 32 bits la System V define lo siguiente:

En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.

En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.
- El valor de retorno será almacenado en EAX.

En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.
- El valor de retorno será almacenado en EAX.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP).

En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.
- El valor de retorno será almacenado en EAX.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH (¿ESP volátil?.)

En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.
- El valor de retorno será almacenado en EAX.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.



En 32 bits la System V define lo siguiente:

- Los registros EBX, EBP, ESI y EDI son **no volátiles**.
- El valor de retorno será almacenado en EAX.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo PUSH debe tener su POP).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo PUSH (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

Para que tengan a mano en la **segunda parte de la materia (32 bits)**:

<b>No volátiles</b>	EBP, EBX, ESI y EDI (¿ESP?)
<b>Valor de retorno</b>	EAX
<b>Parámetros</b>	PUSH a la pila (der. a izq.)
<b>Inv. de pila</b>	Todo PUSH/SUB debe tener su POP/ADD
<b>Llamada a func</b>	Pila alineada a 16 bytes

**Pregunta:** ¿Por qué hace falta alinear la pila a 16 bytes si hacemos una llamada a otra biblioteca?

**Pregunta:** ¿Por qué hace falta alinear la pila a 16 bytes si hacemos una llamada a otra biblioteca?

**Respuesta:** Las funciones pueden hacer uso de operaciones de registros largos (XMM, YMM) que requieren datos alineados a 16 bytes, es por esto que el contrato de uso de un conjunto de instrucciones del procesador se traduce en un contrato de uso de nuestras funciones de bajo nivel.