Ayudantía 4

Subprograms

Javier Larré

Pontificia Universidad Católica de Chile
javier.larre@uc.cl

18 de abril de 2026

Ayudantía de hoy

1. Introducción
2. Resumen de esta semana
3. Ejercicios

Subprograms!

Tarea 2

1. Será de dos lenguajes



Subprograms!

Tarea 2

1. Será de dos lenguajes que tengan funciones de primera clase

Language		Higher-order functions		Nested functions		Non-local variables	
		Arguments	Results	Named	Anonymous	Closures	Partial application
Algol family	ALGOL 60	Yes	No	Yes	No	Downwards	No
	ALGOL 68	Yes	Yes[7]	Yes	Yes	Downwards[7]	No
	Pascal	Yes	No	Yes	No	Downwards	No
	Ada	Yes	No	Yes	No	Downwards	No
	Oberon	Yes	Non-nested only	Yes	No	Downwards	No
	Delphi	Yes	Yes	Yes	2009	2009	No
C family	C	Yes	Yes	Yes in GNU C	Yes in Clang(Blocks)	Yes in Clang(Blocks)	No
	C++	Yes	Yes	C++11[19]	C++11[17]	C++11[21]	C++11
	C#	Yes	Yes	7	2.0 / 3.0	2.0	3.0
	Objective-C	Yes	Yes	Using anonymous	2.0 + Blocks[14]	2.0 + Blocks	No
	Java	Yes	Yes	Using anonymous	Java 8	Java 8	Yes
	Go	Yes	Yes	Using anonymous	Yes	Yes	Yes[18]
	Limbo	Yes	Yes	Yes	Yes	Yes	No
	Newspeak	Yes	Yes	Yes	Yes	Yes	No
	Rust	Yes	Yes	Yes	Yes	Yes	Yes[16]

Figura: Tabla de implementaciones de First Class Functions [b]

[a] First-class function, wikipedia
[b] First-class function, wikipedia

Subprograms!

Tarea 2

1. Será de dos lenguajes que tengan funciones de primera clase
2. Es decir que deberán recibir y retornar funciones

Language		Higher-order functions		Nested functions		Non-local variables	
		Arguments	Results	Named	Anonymous	Closures	Partial application
Algol family	ALGOL 60	Yes	No	Yes	No	Downwards	No
	ALGOL 68	Yes	Yes[7]	Yes	Yes	Downwards[7]	No
	Pascal	Yes	No	Yes	No	Downwards	No
	Ada	Yes	No	Yes	No	Downwards	No
	Oberon	Yes	Non-nested only	Yes	No	Downwards	No
	Delphi	Yes	Yes	Yes	2009	2009	No
C family	C	Yes	Yes	Yes in GNU C	Yes in Clang(Blocks)	Yes in Clang(Blocks)	No
	C++	Yes	Yes	C++11[19]	C++11[17]	C++11[21]	C++11
	C#	Yes	Yes	7	2.0 / 3.0	2.0	3.0
	Objective-C	Yes	Yes	Using anonymous	2.0 + Blocks[14]	2.0 + Blocks	No
	Java	Yes	Yes	Using anonymous	Java 8	Java 8	Yes
	Go	Yes	Yes	Using anonymous	Yes	Yes	Yes[18]
	Limbo	Yes	Yes	Yes	Yes	Yes	No
	Newsqueak	Yes	Yes	Yes	Yes	Yes	No
	Rust	Yes	Yes	Yes	Yes	Yes	Yes[16]

Figura: Tabla de implementaciones de First Class Functions [b]

[a] First-class function, wikipedia
[b] First-class function, wikipedia

Subprograms!

Tarea 2

1. Será de dos lenguajes que tengan funciones de primera clase
2. Es decir que deberán recibir y retornar funciones
3. Subprograms tambien es un tema importante que puede ser preguntado en interrogaciones

Language		Higher-order functions		Nested functions		Non-local variables	
		Arguments	Results	Named	Anonymous	Closures	Partial application
Algol family	ALGOL 60	Yes	No	Yes	No	Downwards	No
	ALGOL 68	Yes	Yes[7]	Yes	Yes	Downwards[7]	No
	Pascal	Yes	No	Yes	No	Downwards	No
	Ada	Yes	No	Yes	No	Downwards	No
	Oberon	Yes	Non-nested only	Yes	No	Downwards	No
	Delphi	Yes	Yes	Yes	2009	2009	No
C family	C	Yes	Yes	Yes in GNU C	Yes in Clang(Blocks)	Yes in Clang(Blocks)	No
	C++	Yes	Yes	C++11[19]	C++11[17]	C++11[21]	C++11
	C#	Yes	Yes	7	2.0 / 3.0	2.0	3.0
	Objective-C	Yes	Yes	Using anonymous	2.0 + Blocks[24]	2.0 + Blocks	No
	Java	Yes	Yes	Using anonymous	Java 8	Java 8	Yes
	Go	Yes	Yes	Using anonymous	Yes	Yes	Yes[18]
	Limbo	Yes	Yes	Yes	Yes	Yes	No
	Newsqueak	Yes	Yes	Yes	Yes	Yes	No
	Rust	Yes	Yes	Yes	Yes	Yes	Yes[14]

Figura: Tabla de implementaciones de First Class Functions [b]

[a] First-class function, wikipedia
[b] First-class function, wikipedia

Subprograms!

Tarea 2

1. Será de dos lenguajes que tengan funciones de primera clase
2. Es decir que deberán recibir y retornar funciones
3. Subprograms tambien es un tema importante que puede ser preguntado en interrogaciones
4. Mezclaremos cosas que ya hemos visto antes



Language		Higher-order functions		Nested functions		Non-local variables	
		Arguments	Results	Named	Anonymous	Closures	Partial application
Algol family	ALGOL 60	Yes	No	Yes	No	Downwards	No
	ALGOL 68	Yes	Yes[7]	Yes	Yes	Downwards[7]	No
	Pascal	Yes	No	Yes	No	Downwards	No
	Ada	Yes	No	Yes	No	Downwards	No
	Oberon	Yes	Non-nested only	Yes	No	Downwards	No
	Delphi	Yes	Yes	Yes	2009	2009	No
C family	C	Yes	Yes	Yes in GNU C	Yes in Clang(Blocks)	Yes in Clang(Blocks)	No
	C++	Yes	Yes	C++11[19]	C++11[17]	C++11[21]	C++11
	C#	Yes	Yes	7	2.0 / 3.0	2.0	3.0
	Objective-C	Yes	Yes	Using anonymous	2.0 + Blocks[14]	2.0 + Blocks	No
	Java	Yes	Yes	Using anonymous	Java 8	Java 8	Yes
	Go	Yes	Yes	Using anonymous	Yes	Yes	Yes[18]
	Limbo	Yes	Yes	Yes	Yes	Yes	No
	Newsqueak	Yes	Yes	Yes	Yes	Yes	No
	Rust	Yes	Yes	Yes	Yes	Yes	Yes[16]

Figura: Tabla de implementaciones de First Class Functions [b]

[a] First-class function, wikipedia
[b] First-class function, wikipedia

Resumen de esta semana

¿Qué es un subprogram?

Resumen de esta semana

Qué es un subprogram?
Es lo que todo este tiempo hemos llamado *funciones* (o procedures)

```
def print_sum(x, y):
    print(f"sum␣is:␣{x␣+␣y}")

print_sum(1, 2)
```

Resumen de esta semana

Qué es un subprogram?
Es lo que todo este tiempo hemos llamado *funciones* (o procedures)

> (...) In a modern programming language, such a collection of statements is written as a subprogram [Sebesta, 2023]

Resumen de esta semana

¿Para qué sirven?

Resumen de esta semana

¿Para qué sirven?

- reuso de espacio, memoria y tiempo escribiendo código.
- Es una abstracción, o bien un nivel de indirección, que sirve para la mejorar legibilidad de nuestro código [Sebesta, 2023] [a]
- Esconden detalles de bajo nivel.

[a] Such reuse is also an abstraction, for the details of the subprogram's computation are replaced in a program by a statement that calls the subprogram.

Resumen de esta semana

¿Qué es un closure?

Resumen de esta semana

¿Qué es un closure?

Es un subprograma (función) junto con el *contexto* en el que fue creado. El contexto son las variables definidas previamente a la definición del subprogram.

Subprograms!

Ejercicios!



Subprograms!

Cuáles son los valores finales de `value` y `list` después de ejecutar con las siguientes estrategias este código escrito con sintáxis *c-like*:

- Call by value
- Call by reference
- Call by value-result

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main() {
    int value = 2, list[5] = {1, 3,
        5, 7, 9};
    swap(value, list[0]);
    swap(list[0], list[1]);
    swap(value, list[value]);
}
```

Subprograms!

	Call by value
value	2
list	1 3 5 7 9

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main() {
    int value = 2, list[5] = {1, 3,
        5, 7, 9};
    swap(value, list[0]);
    swap(list[0], list[1]);
    swap(value, list[value]);
}
```

Subprograms!

	Call by reference
value	2
list	3 1 5 7 9

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main() {
    int value = 2, list[5] = {1, 3,
        5, 7, 9};
    swap(value, list[0]);
    swap(list[0], list[1]);
    swap(value, list[value]);
}
```

Subprograms!

	Call by value-result
value	2
list	3 1 5 7 9

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main() {
    int value = 2, list[5] = {1, 3,
        5, 7, 9};
    swap(value, list[0]);
    swap(list[0], list[1]);
    swap(value, list[value]);
}
```

Subprograms!

Supongamos que queremos un *subprogram* que escribe un header a un archivo y, por cada vez que se llama, escribe el número de veces que se llamó.
Es posible lograr esto **sin variables no locales o parámetros**?

ejercicio.txt

```
HEADER
1
2
```

Subprograms!

Supongamos que queremos un *subprogram* que escribe un header a un archivo y, por cada vez que se llama, escribe el número de veces que se llamó.
Es posible lograr esto **sin variables no locales o parámetros**?
Nope, es imposible: necesitamos una manera de mantener el estado del archivo.
Y si permitimos variables no-locales?

ejercicio.txt

```
HEADER
1
2
```

Subprograms!

ejercicio.py

```
page = 0

def get_closure(path):
    def print_to_file():
        global page
        with open(path, "a") as file:
            if page == 0:
                print("HEADER", file=file)
            else:
                print(page, file=file)
        page += 1

    return print_to_file

closure = get_closure("ejercicio.txt")
closure()
closure()
```

ejercicio.txt

```
HEADER
1
2
```





Ada is a structured, statically typed, imperative, and object-oriented high-level programming language, inspired by Pascal and other languages. Ada was named after Ada Lovelace (1815 - 1852), who has been credited as the first computer programmer.[1]

Los diseñadores le permiten al implementador de Ada 83 escoger la implementación de la keyword `inout` – entre *by copy* o *by reference*

[1] Ada, wikipedia

Los diseñadores le permiten al implementador de Ada 83 escoger la implementación de la keyword `inout` – entre *by copy* o *by reference*
¿Cuál es la ventaja de esta decisión de diseño?

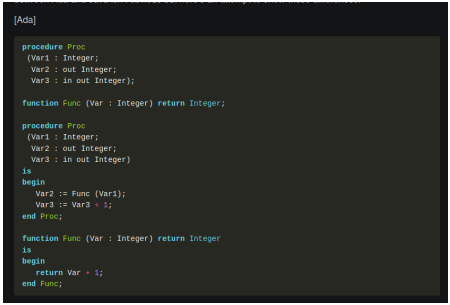


Figura: Ejemplo de *procedures* en Ada [a]

[a] learn.adacore.com

Esto permite flexibilizar las maneras de implementar a Ada. Sin embargo, los implementadores deben tener cuidado de mantener la semántica entre implementaciones.

[Ada]

```ada
procedure Proc
 (Var1 : Integer;
  Var2 : out Integer;
  Var3 : in out Integer);

function Func (Var : Integer) return Integer;

procedure Proc
 (Var1 : Integer;
  Var2 : out Integer;
  Var3 : in out Integer)
is
begin
   Var2 := Func (Var1);
   Var3 := Var3 + 1;
end Proc;

function Func (Var : Integer) return Integer
is
begin
   return Var + 1;
end Func;
```

Figura: Ejemplo de *procedures* en Ada [a]

[a] learn.adacore.com



Fortran is a third-generation, compiled, imperative programming language designed for numeric computation and scientific computing. [2]
En Fortran IV (o por lo menos la mayoría de las implementaciones) los parámetros son pasados por referencia. Señala por lo menos una ventaja y una desventaja sobre esta desición.

[2] Fortran, wikipedia



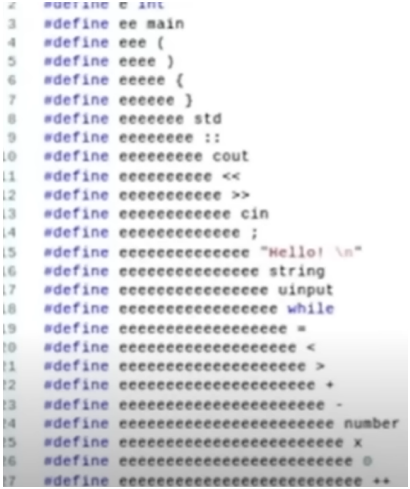


Call by reference permite hacer mutaciones y cambios de estado sin usar operadores (como en el ejemplo que vimos anteriormente). Por otro lado, esto puede introducir fácilmente *bugs* en las aplicaciones. Por esta razón se puede decir que mejora y afecta la legibilidad al mismo tiempo.

Subprograms!

Ejercicios del libro (Capítulo 9)

- Ventajas y desventajas de implementaciones: 1, 2, 3, 14
- Ejecución de código con diferentes estrategias de llamada: 5, 7
- Closures: 4



References


Robert Sebesta (2023)
Concepts of programming languages