Ayudantía 5:
Expresiones y Estructuras de Control: Subprogramas

Profesores: José Luis Martí Lara, Roberto Díaz Urra
Ayudantes:
Hugo Sepúlveda Arriaza
Gabriela Acuña Benito
Lucio Fondón Rebolledo
lucio.fondon@sansano.usm.cl

Universidad Técnica Federico Santa María
Departamento de Informática

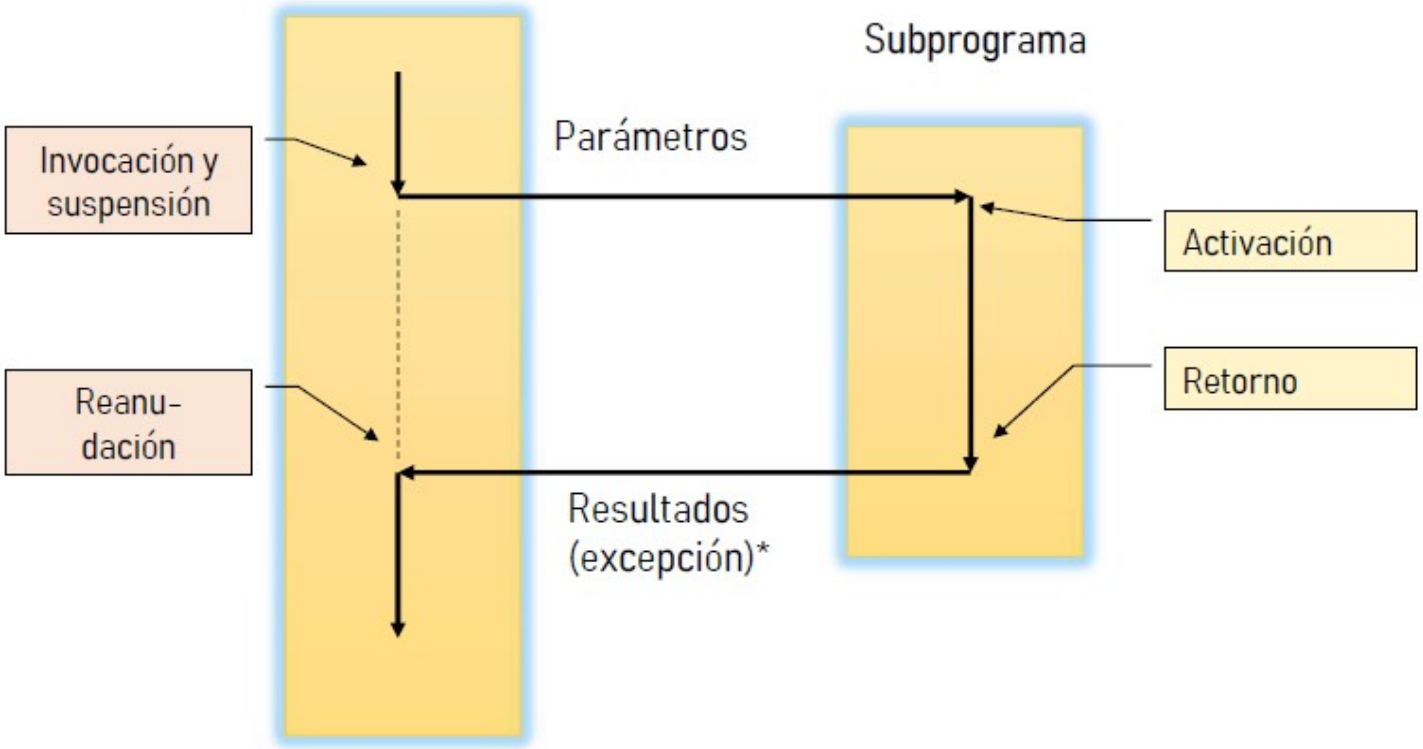
Subprogramas

Definición

Bloques de código que pueden recibir variables (no es completamente necesario), ayudan a reutilizar código (si se implementan correctamente) y normalmente usa memoria dinámica de stack.

- Permite crear *abstracción del proceso* a través de una **interfaz**
- Permite reutilizar código ahorrando memoria y tiempo de codificación
- Existen en forma de procedimientos y subrutinas (no definen un valor de retorno), funciones (definen valor de retorno) y métodos y constructores (en lenguajes orientados a objetos)

Mecanismo de Invocación



Subprogramas: Elementos

Interfaz: Compuesta de cuatro elementos clave:

- **Nombre**: Firma y Protocolo
- **Parámetros**: Parámetros Formales y Comunicación Implícita
- **Valor de Retorno**
- **Excepciones**

```
1  int foo(int a, int b);
2  // Firma -> dos parametros de tipo entero, retorna entero
3  // a,b parametros formales
```

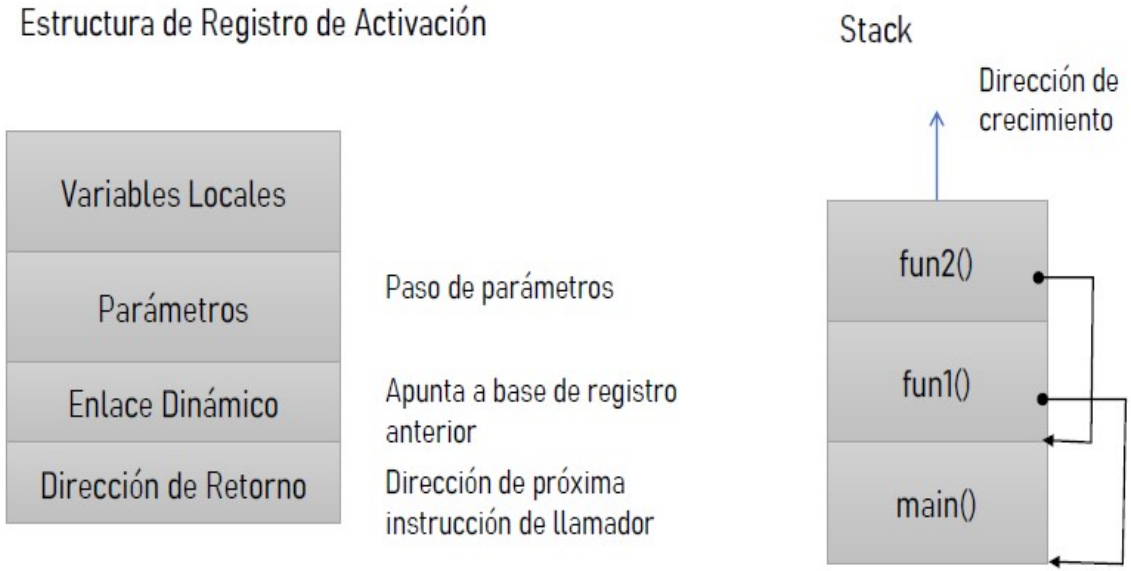
Clases de Valores: Mencionar que los subprogramas pueden ser representados por variables. Según como un lenguaje los usa se pueden definir tres clases:

- **Primera clase**
- **Segunda clase**
- **Tercera clase**

Subprogramas: Estructura y Tipos

Estructura:

- Código: Representa (literal) el código que está escrito para el programa
- Registro de Activación: Representan las variable locales, parámetros y la dirección de retorno del subprograma



Tipos:

- Simple
- Stack

Paso de Parámetros

- **Paso por Valor (IN)**: Se copia el valor de la variable a los parámetros
- **Paso por Resultado (OUT)**: Si los parámetros se modifican, se copia el valor final al parámetro original
- **Paso por Valor-Resultado (INOUT)**: Paso por Valor y resultado juntos.
- **Paso por Referencia**: El parámetro toma la referencia de la variable con la que se invoca.

Paso de Parámetros

- **Paso por Valor (IN)**: Se copia el valor de la variable a los parámetros
- **Paso por Resultado (OUT)**: Si los parámetros se modifican, se copia el valor final al parámetro original
- **Paso por Valor-Resultado (INOUT)**: Paso por Valor y resultado juntos.
- **Paso por Referencia**: El parámetro toma la referencia de la variable con la que se invoca.

```
1 void swap(int a, int b){
2     int t;
3     t   = b;
4     b = a;
5     a = t;
6 }
```

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

¿Cuál función swap realmente intercambia los valores de a y b?

Ejercicios

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. Python y Java pasan todos sus parámetros por referencia
II. Dentro de un registro de activación, es posible encontrar espacio para variables locales y parámetros formales.
III. La firma define la semántica de la interfaz.
IV. El paso por valor es más rápido que el paso por referencia para estructuras de datos de grandes volúmenes.

a) I y IV
b) II, III
c) I, II y III
d) III y IV

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. Python y Java pasan todos sus parámetros por referencia
II. Dentro de un registro de activación, es posible encontrar espacio para variables locales y parámetros formales.
III. La firma define la semántica de la interfaz.
IV. El paso por valor es más rápido que el paso por referencia para estructuras de datos de grandes volúmenes.

a) I y IV
b) II, III
c) I, II y III
d) III y IV

R: **Alternativa b**

Ejercicio 2

```
1  int dato = 2;
2  void algoHace(int a, int b){
3      dato++;
4      a++;
5      b = (a + b) / 2;
6  }
7  int main(){
8      int arreglo [ ] = {1, 3, 5, 7, 9, 11};
9      algoHace(arreglo[1], arreglo[3]);
10     algoHace(dato, arreglo[dato]);
11 }
```

Indicar los valores para `dato` y `arreglo` al analizar el código, si el paso de parámetros es por: valor, referencia y valor-resultado.

Ejercicio 2

```
1  int dato = 2;
2  void algoHace(int a, int b){
3      dato++;
4      a++;
5      b = (a + b) / 2;
6  }
7  int main(){
8      int arreglo [ ] = {1, 3, 5, 7, 9, 11};
9      algoHace(arreglo[1], arreglo[3]);
10     algoHace(dato, arreglo[dato]);
11 }
```

Indicar los valores para `dato` y `arreglo` al analizar el código, si el paso de parámetros es por: valor, referencia y valor-resultado.

R:

- Por valor: `dato = 4`, `arreglo = [1,3,5,7,9,11]`
- Por referencia: `dato = 5`, `arreglo = [1,4,5,5,9,11]`
- Por valor-resultado: `dato = 4`, `arreglo = [1,4,5,4,9,11]`

Ejercicio 3 (C1 2015-1)

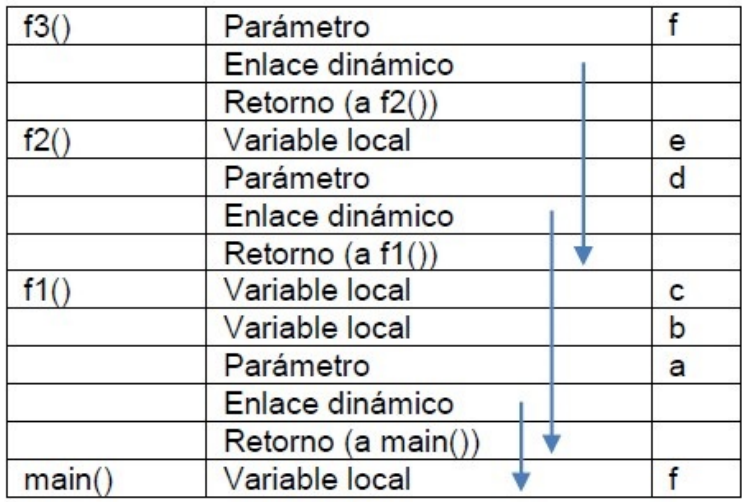
```
1  void f1(float a){
2      int b, c;
3      ...
4      f2(b);
5      ...
6  }
7  void f2(int d){
8      int e;
9      ...
10     f3(e);
11     ...
12 }
13 void f3(int f){
14     ...
15 }
16 int main(){
17     float f;
18     ...
19     f1(f);
20     ...
21 }
```

Mostrar el contenido del stack justo antes de terminar la ejecución de f3().

Ejercicio 3 (C1 2015-1)

```
1  void f1(float a){
2      int b, c;
3      ...
4      f2(b);
5      ...
6  }
7  void f2(int d){
8      int e;
9      ...
10     f3(e);
11     ...
12 }
13 void f3(int f){
14     ...
15 }
16 int main(){
17     float f;
18     ...
19     f1(f);
20     ...
21 }
```

Mostrar el contenido del stack justo antes de terminar la ejecución de f3(). R:



f3()	Parámetro	f
	Enlace dinámico	
	Retorno (a f2())	
f2()	Variable local	e
	Parámetro	d
	Enlace dinámico	
	Retorno (a f1())	
f1()	Variable local	c
	Variable local	b
	Parámetro	a
	Enlace dinámico	
	Retorno (a main())	
main()	Variable local	f