# Ayudantía 2: Continuación de Fundamentos de los Lenguajes de Programación

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

# Variables y Ligados

## Variables

- **Nombre**
- **Dirección (l-value)**
- **Valor (r-value)**
- **Tipo**
- **Tiempo de vida**
- **Ámbito**

## Ligados de Variables

- **Ligado de Tipo**
  - Estático
  - Dinámico
- **Ligado de Memoria**
  - Variables Estática
  - Variables Dinámicas de Stack
  - Variables Dinámicas de Heap
    - Explícitas
    - Implícitas

# Taxonomía de la Memoria

- **Memoria Estática**: Permite realizar ligado estático de una variable a la memoria.
  - Variables globales
  - `static int 1;`
- **Memoria Stack**: Área dinámica de memoria que permite mantener objetos en ambientes de ejecución (bloques, iteradores, funciones, etc.).
  - declaración de variables dentro de bloques
  - `int *ptr; char a;`
- **Memoria Heap**: Permite crear y eliminar objetos en memoria dinámicamente. Son referenciadas indirectamente mediante un puntero o referencia.
  - `int *ptr = (int*)malloc(n * sizeof(int));`

# Ámbitos, Bloques y Anidamientos

## Ámbito

El **ámbito** de una variable es el rango de sentencias en donde una variable puede ser mencionada y usada.

- **Ámbito estático**: Ámbito puede ser determinado antes de la ejecución.
  - Anidados: Definición de variable se busca desde ámbito más cercano al más externo.
  - No anidados: No se permiten subprogramas.
- **Ámbito dinámico**: Ámbito se resuelve por la secuencia de llamada a subprogramas, no por organización del código fuente.
  - Referencias se resuelven por anidamiento de llamadas.

# Ejercicios

# Ejercicio 1

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

I. La memoria del heap permite mantener objetos que se asignan y liberan automáticamente al activar o desactivar un ambiente de ejecución.
II. En ámbitos anidados, la correspondencia entre una referencia a un nombre y su declaración se busca desde el ámbito más interno al más externo.
III. Python y JavaScript son dos lenguajes de programación que hacen uso de variables dinámicas de heap (implícitas), ligando dinámicamente a la memoria del heap cada vez que ocurre una asignación.
IV. Una variable declarada como `int a[20]` dentro de una función en C, corresponde a un arreglo del tipo dinámico de heap.

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. La memoria del heap permite mantener objetos que se asignan y liberan automáticamente al activar o desactivar un ambiente de ejecución.
II. En ámbitos anidados, la correspondencia entre una referencia a un nombre y su declaración se busca desde el ámbito más interno al más externo.
III. Python y JavaScript son dos lenguajes de programación que hacen uso de variables dinámicas de heap (implícitas), ligando dinámicamente a la memoria del heap cada vez que ocurre una asignación.
IV. Una variable declarada como `int a[20]` dentro de una función en C, corresponde a un arreglo del tipo dinámico de heap.

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

**R**: **Alternativa b**

# Ejercicio 2 (C1 2016-1)

```
x = 1
y = 3
z = 5
def sub1():
    a = 7
    y = 9
    z = 11
    ...
def sub2():
    global x
    a = 13
    x = 15
    w = 17
    ...
    def sub3():
        nonlocal a
        a = 19
        b = 21
        z = 23
        ...
```

Considerar el siguiente código en Python, liste todas las variables, junto con el subprograma en donde ellas son declaradas, que son visibles en los cuerpos de sub1(), sub2() y sub3(), asumiendo ámbito estático:

# Ejercicio 2 (C1 2016-1)

```
x = 1
y = 3
z = 5
def sub1():
    a = 7
    y = 9
    z = 11
    ...
def sub2():
    global x
    a = 13
    x = 15
    w = 17
    ...
    def sub3():
        nonlocal a
        a = 19
        b = 21
        z = 23
        ...
```

Considerar el siguiente código en Python, liste todas las variables, junto con el subprograma en donde ellas son declaradas, que son visibles en los cuerpos de sub1(), sub2() y sub3(), asumiendo ámbito estático:

- `sub1()`
  - a,y,z: variable local
  - x: variable global
- `sub2()`
  - a,w: variable local
  - x,y.z: variable global
- `sub3()`
  - a,x,w: variable de `sub2()`
  - b,z: variable local
  - y: variable global

# Ejercicio 3

```c
int Foo(int* p){
    static int time = 0;
    time++;
    return *p * time;
}

int main(){
    int x = 3;
    int y = 4;
    int n = Foo(&x)*Foo(&y);
    printf("%d\n",n);
    return 0;
}
```

Dado el siguiente código en C, indique lo que muestra por pantalla:

# Ejercicio 3

```c
int Foo(int* p){
    static int time = 0;
    time++;
    return *p * time;
}

int main(){
    int x = 3;
    int y = 4;
    int n = Foo(&x)*Foo(&y);
    printf("%d\n",n);
    return 0;
}
```

Dado el siguiente código en C, indique lo que muestra por pantalla:

R:

<< 24

# Ejercicio 4 (Q2 2018-2)

```
int a,*b,c = 2;

void Foo(){
    int *n, a;
    a = 2;
    n = &a;
    //Break
}

int main(){
    int c = 1, *d;
    d = (int*)malloc(sizeof(int));
    b = &a;
    *b = 4;
    *d = c;
    Foo();
    //...
}
```

Muestre el ligado de memoria y el valor de cada variable cuando el siguiente código en C pasa por la sentencia "//Break"

# Ejercicio 4 (Q2 2018-2)

```
int a,*b,c = 2;

void Foo(){
    int *n, a;
    a = 2;
    n = &a;
    //Break
}

int main(){
    int c = 1, *d;
    d = (int*)malloc(sizeof(int));
    b = &a;
    *b = 4;
    *d = c;
    Foo();
    //...
}
```

Muestre el ligado de memoria y el valor de cada variable cuando el siguiente código en C pasa por la sentencia "//Break" R:

- **Estática:**
  - a = 4;
  - *b = 4;
  - c = 2;
- **Stack:**
  - c = 1;
  - d= Dir de memoria;
  - n = Dir de memoria;
  - *n = 2;
  - a = 2;
- **Heap:**
  - *d = 1;