Capítulo 1 - *Algoritmos*

1.5.3 – *Estructuras selectivas anidadas*

- Las estructuras selectivas **si** y **si-sino** implican la selección de una de dos alternativas. Es posible utilizar la instrucción **si** para diseñar estructuras de selección que contengan más de dos alternativas.

```
si <condicion 1>
  si <condicion 2>
    <accion 1>
    <accion 2>

    <accion n>
  fin_si
fin_si
```

```
si <condicion 1>
  si <condicion 2>
    <accion 1>

    <accion n>
  sino
    <accion 1>

    <accion n>
  fin_si
sino
 si <condicion 2>
    <accion 1>

    <accion n>
  sino
    <accion 1>

    <accion n>
  fin_si
fin_si
```


Capítulo 1 - *Algoritmos*

1.5.3 – *Estructuras repetitivas anidadas*

- De igual forma que se pueden anidar estructuras se selección, es posible insertar un bucle dentro de otro

```
mientras <condicion1>
  mientras <condicion2>
    <accion 1>
    <accion 2>

    <accion n>
  fin_mientras
fin_mientras
```

```
desde v<-vi hasta vf
  desde v<-vi hasta vf
    <accion 1>
    <accion 2>

    <accion n>
  fin_desde
fin_desde
```

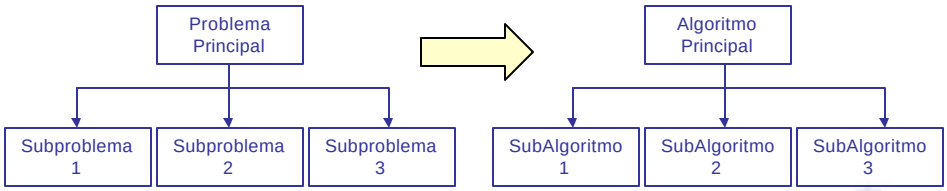
```
repetir
  repetir
    <accion 1>
    <accion 2>

    <accion n>
  hasta <condicion2>
hasta <condicion1>
```

Capítulo 1 - *Algoritmos*

1.6 – *Subprogramas (Subalgoritmos)*

- Un método conocido para solucionar un problema complejo es dividirlo en subproblemas, o problemas más sencillos. La misma técnica se puede aplicar a los algoritmos, donde a cada uno de estos subproblemas le llamaremos **Subprogramas o Subalgoritmos.**



Capítulo 1 - *Algoritmos*

1.6.1 – *Funciones*

- Una **función** es una operación que toma uno o más valores llamados **argumentos** y produce un valor denominado **resultado**.

```
<tipo_de_resultado> funcion <nombre_fun> (lista de parametros)
[declaraciones locales]
inicio
  <acciones>
  devolver (<expresion>)
fin_funcion
```

- Los argumentos de la declaración de la función se denominan parámetros formales y sólo se utilizan dentro del cuerpo de la función.

```
real funcion f1(real:x)
real: y
inicio
  y<-x/(1+(x*x))
  devolver (y)
fin_funcion
```

Capítulo 1 - *Algoritmos*

1.6.1.1 – *Invocación a las Funciones*

- Una función se llama de la siguiente forma

```
nombre_funcion (lista de parametros actuales)
```

```
s <- f1(r)
```

- Los argumentos utilizados en la llamada a la función se denominan parámetros actuales.
- Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los parámetros formales y los actuales.
- Debe haber exactamente el mismo número de parámetros actuales que de parámetros formales en la declaración de la función y se presupone correspondencia uno a uno de izquierda a derecha entre los parámetros formales y los actuales.

Capítulo 1 - *Algoritmos*

1.6.2 – *Procedimientos*

- Con frecuencia , se requieren subprogramas que entreguen varios resultados en vez de uno, o por ejemplo que realicen la ordenación de una serie de números. En estas situaciones la función no es apropiada y se necesita disponer de otro tipo de subprograma: el **procedimiento o subrutina**.

```
procedimiento <nombre_proc> (lista de parametros formales)
[declaraciones locales]
inicio
  <acciones>
fin_procedimiento
```

```
procedimiento p1(E real:x,E real:y,S real:mult, S real:sum)
inicio
  sum<-x+y
  mult<-x*y
fin_procedimiento
```

Capítulo 1 - *Algoritmos*

1.6.2 – *Invocación a los Procedimientos*

- Un procedimiento se llama de la siguiente forma

```
llamar_a nombre_procedimiento(lista de parametros actuales)
```

```
llamar_a p1(m,n,r,t)
```

- En los procedimientos los parámetros actuales y formales tienen el mismo significado que en las funciones.
- Cada vez que se llama a un procedimiento desde el algoritmo principal se establece automáticamente una correspondencia entre los parámetros formales y los actuales.
- Debe haber exactamente el mismo número de parámetros actuales que de parámetros formales en la declaración del procedimiento y se presupone correspondencia uno a uno de izquierda a derecha entre los parámetros formales y los actuales.