Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde

FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N°8

Estructuras de Control: Iterativas

RESUMEN DE CONTENIDOS N° 8

Estructuras de Control: Iterativas

Introducción

Analizaremos ahora las diferentes formas de plantear, en la construcción de algoritmos, la repetición de la ejecución de una o varias acciones, a través de las estructuras de Iteración.

Las estructuras de iteración son aquellas que nos permiten ejecutar una acción o un conjunto de acciones varias veces, dependiendo de una condición.

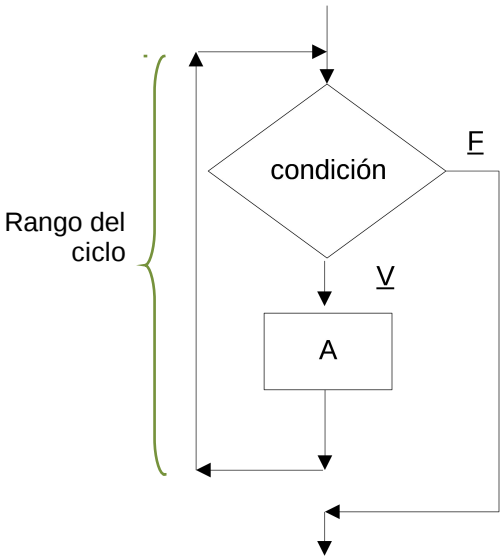
En la solución de problemas computacionales, se presentan casos en los que no se conoce de antemano la cantidad de veces que se quiere repetir un conjunto de acciones, en otros, ese número de repeticiones es conocido. Definiremos estructuras algorítmicas que nos permitan describir ambas situaciones.

Dentro de las estructuras de iteración, veremos tres esquemas básicos de la programación estructurada en pseudocódigo y luego su equivalente en el lenguaje C++:

➢ **MIENTRAS**	➢ **WHILE**
➢ **REPETIR**	➢ **DO-WHILE**
➢ **PARA**	➢ **FOR**

MIENTRAS y WHILE

La estructura **Mientras**, se representa como:
En diagrama de flujo:



donde **condición** puede ser una variable o expresión lógica y **A** representa cualquier acción primitiva o conjunto de estructuras, llamado **rango del ciclo**.
Como en las estructuras vistas anteriormente -secuencia y selección- la iteración se considera una unidad, con un sólo punto de entrada y un sólo punto de salida.

En pseudocódigo la sintaxis equivalente al diagrama es:

MIENTRAS condición **HACER**

A

FINMIENTRAS

Las palabras claves **MIENTRAS** y **FINMIENTRAS** identifican, respectivamente, el principio y fin de la estructura.

Cuando el procesador encuentra esta estructura deberá:

- Observar la condición
- Si la condición es **V** ⇒ ejecutar **A**
- Observar nuevamente la condición
- Si es **V** , ejecutar **A** y volver a evaluar la condición y así sucesivamente hasta que de la observación de la condición se obtenga un valor **falso**
- Si es **F,** finaliza la ejecución
- Si la condición es **F** ⇒ salir de la estructura

Debemos notar que:

- El bloque A puede no llegar a ejecutarse, si al principio, la condición es falsa.
- Entre el conjunto de acciones de A, deberá existir una acción que permita modificar, en determinado momento, su valor de verdad; sino el bloque A se repetirá indefinidamente.
- No es necesario conocer de antemano el número de repeticiones, o bien no es necesario expresarlo en forma directa.
- En la estructura Mientras el bloque A pueda ser ejecutado desde 0 a n veces.

En C++, esta estructura es equivalente al **While**. Las acciones abarcadas por esta estructura se ejecutan repetidamente mientras la expresión lógica arroje un valor **distinto de cero** (verdadero).

Sintaxis	Ejemplo

`while (expresión lógica )`
`{`
`  acciones`
`}`
	`int a=0;`
	`while ( a<100 )`
	`{`
	`  cout << a<< "\n";`
	`  a++;`
	`}`

Ejemplo Práctico 1:

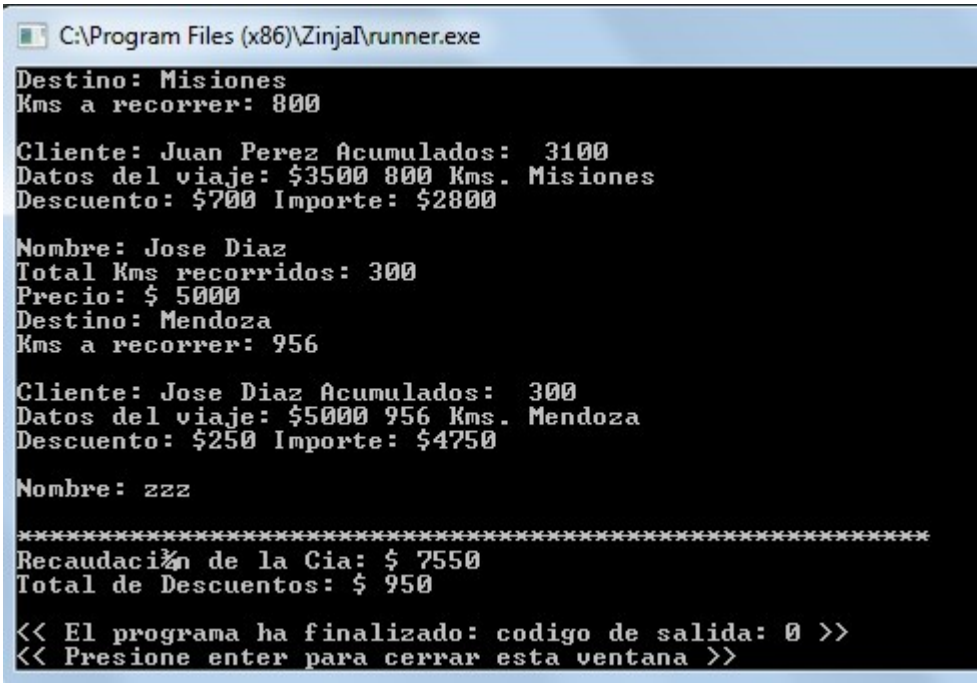
Una compañía de turismo ha definido una política de promoción de sus empresas para lo cual ha fijado descuentos para sus clientes en función de los viajes anteriores realizados. El valor del descuento es del 20% para aquellos clientes que, considerando los viajes realizados en el último año y el que desean realizar, han recorrido más de 3000 Kms. y del 5% para aquellos que no han alcanzado dicha cifra.
Se desea realizar un algoritmo que calcule la recaudación de la compañía en un día. Para ello, por cada cliente se ingresa su nombre y apellido, el total de Kms. recorridos hasta el momento, y los datos del viaje a contratar: precio, destino, Kms. a recorrer. El fin de datos se produce al ingresar como nombre y apellido un valor "ZZZ" .
Para cada cliente informar los datos ingresados, el monto del descuento y el monto a pagar. Informar además el total recaudado y el total de descuentos realizados por la Compañía, con leyendas alusivas.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string nya, destino;
    int kms, kmrec, total;
    float precio, desc, imp;
    float totdesc=0;
    float totrec=0;

    cout<<"Ingrese los siguientes datos:"<<endl;
    cout<<"Nombre: ";
    getline(cin,nya);
    while(nya!="zzz")
    {
        cout<<"Total Kms recorridos: ";
        cin>>kmrec;
        cout<<"Precio: $ ";
        cin>>precio;
        cin.get();
        cout<<"Destino: ";
        getline(cin,destino);
        cout<<"Kms a recorrer: ";
        cin>>kms;
        total=kmrec+kms;
        if(total>3000)
            desc=precio*0.20;
        else desc=precio*0.05;
        imp=precio-desc;

        cout<<endl;
        cout<<"Cliente: "<<nya<<" Acumulados:  "<<kmrec<<endl;
        cout<<"Datos del viaje: $"<<precio<<" "<<kms<<" Kms."<<" "<<destino<<endl;
        cout<<"Descuento: $"<<desc<<" Importe: $"<<imp<<endl;
        cout<<endl;
        totdesc=totdesc+desc;
        totrec=totrec+imp;
        cout<<"Nombre: ";
        cin.get();
        getline(cin,nya);
    }
    cout<<endl; //linea en blanco
    cout<<"*****************************************************"<<endl;
    cout<<"Recaudación de la Cia: $ "<<totrec<<endl;
    cout<<"Total de Descuentos: $ "<<totdesc;
    return 0;
}
```

Seguimiento:

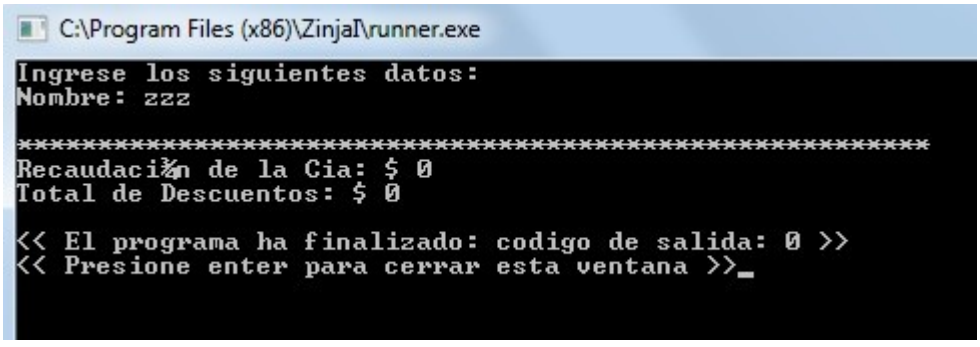
a) Datos: “Juan Pérez”, 3100, 3500, “Misiones”, 800
“José Díaz”, 300, 5000, “Mendoza”, 956
“ZZZ”

NYA	KMREC	PRECIO	KMS	DEST	TOTAL	DESC	IMP	TOTDES	TOTREC
“Juan Pérez”	3100	3500	800	“Misiones”	3900	700	2800	0	0
								700	2800
“José Díaz”	300	5000	956	“Mendoza”	1256	250	4750	950	7550
“ZZZ”									



b) Datos: “ZZZ”

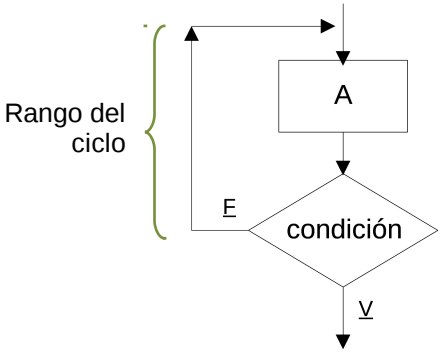
NYA	KMREC	PRECIO	KMS	DEST	TOTAL	DESC	IMP	TOTDES	TOTREC
“ZZZ”								0	0



Observación: debemos notar que en este ejemplo se desconoce a priori la cantidad de veces que se repite el proceso. Sin embargo, el mismo algoritmo puede utilizarse para el caso de `ningún’ cliente (ejemplo b), para un solo cliente, para 2 (ejemplo a) o para **n** clientes.

REPETIR y DO-WHILE

La estructura **Repetir**, se representa como:
En diagrama de flujo:



donde **condición** puede ser una variable o expresión lógica y **A** representa cualquier acción primitiva o conjunto de estructuras, llamado **rango del ciclo**.
.

En pseudocódigo la sintaxis equivalente al diagrama es:

```
REPETIR
        A
HASTAQUE condición
```

donde el principio y el fin de la estructura están dados por las palabra claves **REPETIR** y **HASTAQUE.**

Tal como en el caso anterior, esta estructura es considerada como una unidad, con un único punto de entrada y un único punto de salida.

Esta estructura plantea la ejecución del bloque **A**, reiteradamente, hasta que se cumpla la condición.

En presencia de este esquema, el procesador deberá:

- ejecutar el bloque **A**
- Observar la **condición**
- Si la condición es **F** ⇒ ejecutar **A**
- Observar nuevamente la **condición**
- Si es **F** , ejecutar **A** y vuelve a evaluar la condición y así sucesivamente hasta que de la observación de la condición se obtenga un valor verdadero
- Si es **V** , sale de la estructura
- Si la condición es **V** ⇒ finaliza la ejecución de la estructura

Debemos notar que:

- La condición es evaluada después de cada ejecución del bloque A, por lo tanto éste será ejecutado al menos una vez.

- Entre el conjunto de acciones de A, deberá existir una acción que permita modificar, en determinado momento, su valor de verdad; sino el bloque A se repetirá indefinidamente.
- La diferencia entre la estructura Repetir y la estructura Mientras, reside en el punto de evaluación de la condición. Esto hace que en la estructura Mientras el bloque A pueda ser ejecutado desde 0 a n veces, mientras que en el Repetir desde 1 a n veces.

En C++, esta estructura es equivalente al **Do-While**. Las acciones abarcadas por esta estructura se ejecutan repetidamente hasta que la expresión lógica arroje el resultado **cero** (falso).

Sintaxis	Ejemplo

`Do`
`{`
`  acciones`
`} while (expresión lógica );`
	`int b=0;`
	`do`
	`{`
	`  b++ ;`
	`  cout << b<< "\n" ;`
	`}  while ( b<100 );`

Ejemplo Práctico 2:
Una compañía de turismo ha definido una política de promoción de sus empresas para lo cual ha fijado descuentos para sus clientes en función de los viajes anteriores realizados. El valor del descuento es del 20% para aquellos clientes que, considerando los viajes realizados en el último año y el que desean realizar, han recorrido más de 3000 Kms. y del 5% para aquellos que no han alcanzado dicha cifra.
Se desea definir un algoritmo que calcule la recaudación diaria de la compañía para sus N clientes. Para ello, por cada cliente se ingresa su nombre y apellido, el total de Kms. recorridos hasta el momento, y los datos del viaje a contratar: precio, destino, Kms. a recorrer. Para cada cliente informar los datos ingresados, el monto del descuento y el monto a pagar. Informar además el total recaudado y el total de descuentos efectuados por la Compañía, con leyendas alusivas.
El valor N se ingresa como primer dato.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string nya, destino;
    int kms, kmrec, total;
    float precio, desc, imp;
    float totdesc=0;
    float totrec=0;
    int cont=0;
    int N;

    cout<<"Ingrese cantidad de clientes a cargar: ";
    cin>>N;
    cout<<endl;
    cout<<"Ingrese los siguientes datos:"<<endl;
    cin.get();
    do
    {
        cout<<"Nombre: ";
        getline(cin,nya);
        cout<<"Total Kms recorridos: ";
        cin>>kmrec;
        cout<<"Precio: $ ";
        cin>>precio;
        cin.get();
        cout<<"Destino: ";
        getline(cin,destino);
        cout<<"Kms a recorrer: ";
        cin>>kms;
        total=kmrec+kms;
        if(total>3000)
            desc=precio*0.20;
        else desc=precio*0.05;
        imp=precio-desc;
        cout<<endl;
        cout<<"Cliente: "<<nya<<" Acumulados:  "<<kmrec<<endl;
        cout<<"Datos del viaje: $"<<precio<<" "<<kms<<" Kms."<<" "<<destino<<endl;
        cout<<"Descuento: $"<<desc<<" Importe: $"<<imp<<endl;
        cout<<endl;
        totdesc=totdesc+desc;
        totrec=totrec+imp;
        cont++;
        cin.get();
    }while(cont<N);
    cout<<endl;
    cout<<"************************************************************"<<endl;
    cout<<"Recaudación de la Cia: $ "<<totrec<<endl;
    cout<<"Total de Descuentos: $ "<<totdesc;
    return 0;
}
```

Seguimiento:

Datos: 2, “Juan Pérez”, 3100, 3500, “Misiones”, 800
“José Díaz”, 300, 5000, “Mendoza”, 956

NYA	KMREC	PRECIO	KMS	DEST	TOTAL	DESC	IMP	TOTDES	TOTREC	N	CONT
“Juan Pérez”	3100	3500	800	“Misiones”	3900	700	2800	0	0	2	0
								700	2800		1
“José Díaz”	300	5000	956	“Mendoza”	1256	250	4750	950	7550		2

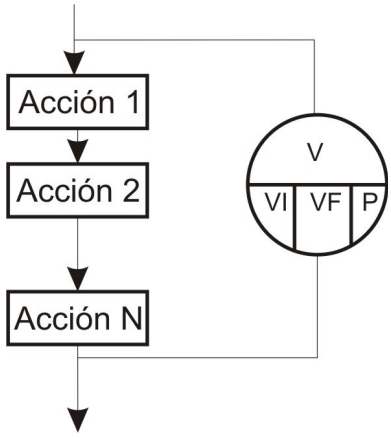
```
Ingrese cantidad de clientes a cargar: 2

Ingrese los siguientes datos:
Nombre: Juan Perez
Total Kms recorridos: 3100
Precio: $ 3500
Destino: Misiones
Kms a recorrer: 800

Cliente: Juan Perez Acumulados:  3100
Datos del viaje: $3500 800 Kms. Misiones
Descuento: $700 Importe: $2800

Nombre: Jose Diaz
Total Kms recorridos: 300
Precio: $ 5000
Destino: Mendoza
Kms a recorrer: 956

Cliente: Jose Diaz Acumulados:  300
Datos del viaje: $5000 956 Kms. Mendoza
Descuento: $250 Importe: $4750


****************************************************************
Recaudación de la Cia: $ 7550
Total de Descuentos: $ 950

<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

Observación: vemos en este ejemplo que conocemos de antemano la cantidad de clientes a procesar a través de la variable N.

PARA y FOR

La estructura de repetición llamada **PARA** nos permite realizar un conjunto de acciones un número determinado de veces.

En diagrama de flujo:



Donde:

V : es una variable numérica llamada variable de control

VI: es variable o constante numérica, es el valor inicial que toma V

VF: variable o constante numérica, es el valor final que toma V

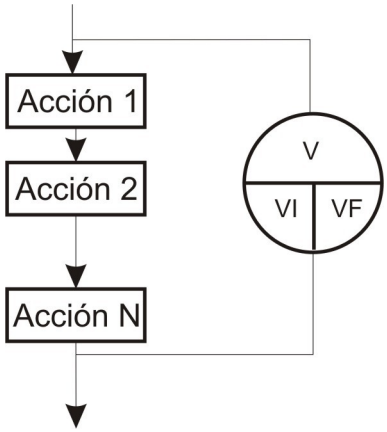
P : variable o constante numérica, es el paso, el valor de P puede ser positivo o negativo pero no puede ser cero.

A diferencia de las estructuras REPETIR y MIENTRAS, en esta estructura iterativa el procesador tiene a su cargo la variación de la variable de control ya que se incrementa su valor automáticamente de acuerdo a P, hasta llegar a VF.

El procesador ejecutará las acciones en el siguiente orden:

1. A la variable V le asigna el valor de VI.
2. Ejecuta las acciones previstas (acción 1 hasta acción n).
3. Incrementa la variable V sumándole P.
4. Evalúa si el valor de la variable V de control es menor o igual a VF en ese caso continúa a partir de 2.
5. Si la variable V es mayor a VF finaliza la ejecución de la estructura y continúa con la ejecución del algoritmo.

Cuando P = 1 el formato se reduce a:



Observaciones:

1- Cuando P > 0 el valor de VF debe ser mayor o igual a VI.

2- Cuando P < 0 el valor de VF debe ser menor o igual a VI

3- Las acciones 1 a n se ejecutan siempre al menos una vez

En pseudocódigo la sintaxis equivalente al diagrama es:

```
Para I desde VI hasta VF con Paso P Hacer
        A
FinPara
```

donde el principio y el fin de la estructura están dados por las palabra claves **PARA** y **FINPARA.**
Tal como en el caso anterior, esta estructura es considerada como una unidad, con un único

En C++, esta estructura es equivalente al **For**. Las acciones abarcadas por esta estructura se ejecutan repetidamente hasta que la **exp2** arroje **cero** (falso); **exp1** es una expresión de inicialización y se ejecutan una única vez; **exp3** se realiza al final del grupo de acciones y generalmente se emplea para incrementar la variable que controla la estructura.

Sintaxis	Ejemplo

`for (exp1; exp2; exp3)`
`{`
`  acciones`
`}`
	`int a=0;`
	`for ( a=0 ; a<100 ; a++ )`
	`  cout << a<< "\n" ;`

Ejemplo Práctico 3:
Una compañía de turismo ha definido una política de promoción de sus empresas para lo cual ha fijado descuentos para sus clientes en función de los viajes anteriores realizados. El valor del descuento es del 20% para aquellos clientes que, considerando los viajes realizados en el último año y el que desean realizar, han recorrido más de 3000 Kms. y del 5% para aquellos que no han alcanzado dicha cifra.
Se desea definir un algoritmo que calcule la recaudación diaria de la compañía para sus N clientes. Para ello, por cada cliente se ingresa su nombre y apellido, el total de Kms. recorridos hasta el momento, y los datos del viaje a contratar: precio, destino, Kms. a recorrer. Para cada cliente informar los datos ingresados, el monto del descuento y el monto a pagar. Informar además el total recaudado y el total de descuentos efectuados por la Compañía, con leyendas alusivas.
El valor N se ingresa como primer dato.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string nya, destino;
    int kms, kmrec, total;
    float precio, desc, imp;
    float totdesc=0;
    float totrec=0;
    int i;
    int N;

    cout<<"Ingrese cantidad de clientes a cargar: ";
    cin>>N;
    cout<<endl;
    cout<<"Ingrese los siguientes datos:"<<endl;
    cin.get();
    for(i=0;i<N;i++)
    {
        cout<<"Nombre: ";
        getline(cin,nya);
        cout<<"Total Kms recorridos: ";
        cin>>kmrec;
        cout<<"Precio: $ ";
        cin>>precio;
        cin.get();
        cout<<"Destino: ";
        getline(cin,destino);
        cout<<"Kms a recorrer: ";
        cin>>kms;
```

```
32          total=kmrec+kms;
33          if(total>3000)
34              desc=precio*0.20;
35          else desc=precio*0.05;
36          imp=precio-desc;
37          cout<<endl;
38          cout<<"Cliente: "<<nya<<" Acumulados:  "<<kmrec<<endl;
39          cout<<"Datos del viaje: $"<<precio<<" "<<kms<<" Kms."<<" "<<destino<<endl;
40          cout<<"Descuento: $"<<desc<<" Importe: $"<<imp<<endl;
41          cout<<endl;
42          totdesc=totdesc+desc;
43          totrec=totrec+imp;
44          cin.get();
45      };
46      cout<<endl;
47      cout<<"*************************************************************"<<endl;
48      cout<<"Recaudación de la Cia: $ "<<totrec<<endl;
49      cout<<"Total de Descuentos: $ "<<totdesc;
50      return 0;
51  }
52
```

Seguimiento:

Datos: 2, “Juan Pérez”, 3100, 3500, “Misiones”, 800
“José Díaz”, 300, 5000, “Mendoza”, 956

```
Ingrese cantidad de clientes a cargar: 2

Ingrese los siguientes datos:
Nombre: Juan Perez
Total Kms recorridos: 3100
Precio: $ 3500
Destino: Misiones
Kms a recorrer: 800

Cliente: Juan Perez Acumulados:  3100
Datos del viaje: $3500 800 Kms. Misiones
Descuento: $700 Importe: $2800

Nombre: Jose Diaz
Total Kms recorridos: 300
Precio: $ 5000
Destino: Mendoza
Kms a recorrer: 956

Cliente: Jose Diaz Acumulados:  300
Datos del viaje: $5000 956 Kms. Mendoza
Descuento: $250 Importe: $4750


*************************************************************
Recaudación de la Cia: $ 7550
Total de Descuentos: $ 950

<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

El operador coma y la sentencia for

C++ permite a través del operador coma (**,**) realizar más de una instrucción donde generalmente se admite una.

Por ejemplo, en el ciclo for, la primera expresión es usada comúnmente para inicializar una variable y la tercera expresión para modificar la variable que controla la estructura. Empleando el operador coma, puede efectuarse más de una inicialización e incremento (o decremento) de las variables inicializadas.

Ejemplo

`int i, j;`
`for (i=0, j=10; i < 10 ; i++, j--)`
`  cout << i << " " << j << endl;`

SALTO NO CONDICIONAL O INTERRUPCIÓN (Break y Continue)

Ambas sentencias interrumpen la ejecución del grupo de acciones abarcadas por una estructura repetitiva, saltando al final de la estructura.
Luego de la interrupción, **break**, continúa con la sentencia que sigue a la iteración, abandonando la estructura de repetición; **continue** en cambio, salta al final de la estructura de repetición pero no la abandona, y permite continuar con la próxima iteración.

Ejemplo de `break`	Ejemplo de `continue`

`int a=0;`
`while (a<5)`
`{`
`  a++;`
`  if a == 4 break;`
`  cout << a;`
`}`
	`int a=0;`
	`while (a<5)`
	`{`
	`  a++;`
	`  if a == 4 continue;`
	`  cout << a;`
	`}`

Salida: 1 2 3	Salida: 1 2 3 5

ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

En el diseño de algoritmos, es común la utilización de estructuras lógicas de control complejas, las que se basan en la combinación de estructuras elementales.
Por ejemplo, los **condicionales anidados**, constituyen una estructura de control compleja, dado que es una estructura de decisión dentro de otra estructura de decisión. De la misma manera, los **ciclos anidados**, pues son estructuras de iteración incluidas dentro de otras siguiendo las mismas reglas: la estructura interna debe estar totalmente incluida en la estructura externa.

A continuación, se presentan algunos ejemplos:

Condicionales anidados

En pseudicódigo	En C++

`SI condicion 1`
 	
`  ENTONCES A`
`  SI condicion 2`
 	
`  ENTONCES B`
 	
`  SINO`
 	
`  C; D`
`  FINSI`
	`if (condicion 1)`
	`{`
	`  A;`
	`  if (condicion 2)`
	`  {`
	`  B;`
	`  }`
	`  else`
	`  {`
	`  C; D;`
	`  }`
	`}`

```
   SINO
                                          SI  condicion 3
                                              ENTONCES M
                                                SINO
                                                   R
                                           FINSI
                                           W
FINSI
```

```
else
  {
      if ( condicion 3)
        {
         M;
        }
        else
            {
             R;
             }
        W;
  }
```

Condicionales anidados, dentro de un ciclo iterativo

En pseudicódigo	En C++

`MIENTRAS condicion 1  HACER`
 	
`  SI  condicion 2`
`    ENTONCES`
`           SI  condicion 3`
 	
`                    ENTONCES B`
 	
`           FINSI`
`    SINO`
 	
`                  A`
`  FINSI`
`FINMIENTRAS`
	`while ( condicion 1 )`
	`{`
	`    if ( condicion 2)`
	`       {`
	`        if ( condicion 3 )`
	`            {`
	`             B;`
	`            }`
	`       }`
	`        else`
	`            {`
	`             A;`
	`            }`
	`}`

Ciclos iterativos Para anidados

En pseudicódigo	En C++

`Para I desde 1 hasta 10 hacer`
 	
`  A`
`  Para J desde 1 hasta 5 hacer`
 	
`    B;`
`    C`
`  Finpara`
`Finpara`
	`for ( i=1 ; i<11 ; i++ )`
	` {`
	`  A;`
	`  for ( j=1 ; j<6 ; j++ )`
	`   {`
	`    B;`
	`    C;`
	`   }`
	`}`

La función exit()

Esta función del lenguaje C++ permite interrumpir un programa, devolviendo un valor al entorno o plataforma empleado (DOS, UNIX, LINUX).
Se halla definida en **stdlib.h**, por lo cual, si se desea utilizar esta función, deberá incluirse este archivo en la cabecera del programa. La función, devuelve el valor de su argumento:
void exit(int)
El valor entero que se indica como argumento se retorna al proceso padre que invocó al programa. Los valores que devuelve pueden ser diferentes, pero un valor igual a cero, indica que la interrupción del programa se ha efectuado con éxito. Un valor entero distinto de cero, indica que la interrupción del programa se ha debido a un error.

Ejemplo

```
cout<<"Desea continuar operando con el programa (S/N)?";
cin >> resp ;
resp = toupper( resp );   // pasa a mayúsculas
if  (resp=='S') exit(0);
```

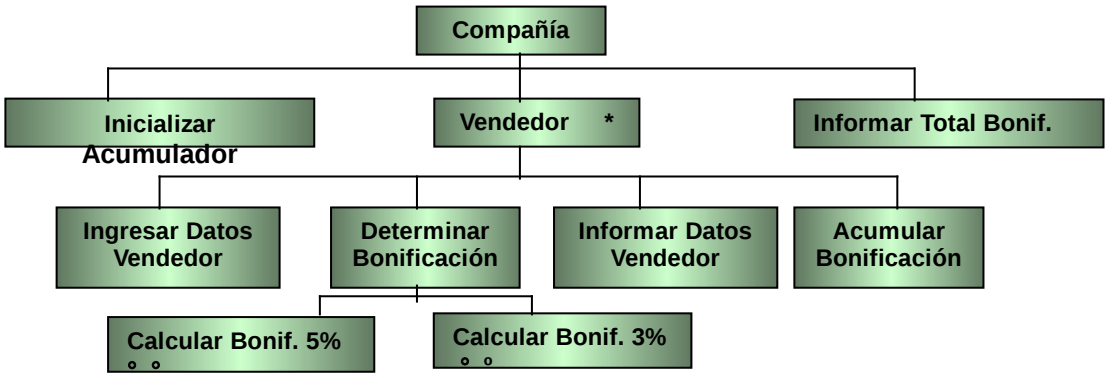
ESTRATEGIAS

En la unidad 1, definimos ESTRATEGIA y ubicamos a la **Definición de una estrategia** como una etapa dentro de la **Resolución de problemas**, que debe realizarse previa a la **Definición de algoritmos**.
Comenzaremos a aplicar estos conceptos y los vistos en esa unidad referidos a diseño descendente, en la resolución de problemas que se nos plantean.

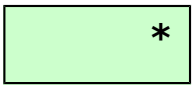
Ejemplo Práctico 4:
Una compañía paga a cada uno de sus vendedores una bonificación anual basada en el sueldo del vendedor y en su total de ventas durante el año. El valor de la bonificación es del 3 % del total de las ventas si las ventas son inferiores a 4 veces el sueldo del vendedor, o del 5 % si son mayores o iguales a ese monto.
Realizar un algoritmo que ingrese por cada vendedor el nombre y apellido, el sueldo y el total de ventas. El fin de datos se produce al ingresar como nombre y apellido 'ZZZZ'.
Para cada empleado informar el nombre y apellido y la bonificación correspondiente, y además informar el total que debe pagar la compañía en concepto de bonificaciones, con leyenda indicativa.

Para resolver este problema, luego de realizar el análisis del mismo, debemos plantear la estrategia, definiendo las etapas que deben seguirse para arribar a la solución del problema, indicando **QUÉ** debe hacerse y no el **CÓMO.**

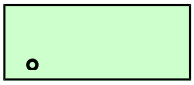
En el enunciado planteado una alternativa de Estrategia es:



Notación:



Representa un proceso iterativo. Debajo de él pueden graficarse los módulos que forman el ciclo.



Representa una alternativa cuya ejecución dependerá de una condición.