- ✓ **Estructura de Control**: Construcción de un lenguaje de programación que permite alterar el flujo secuencial en el que por defecto se ejecutan las sentencias de un programa

- ✓ **Programación Estructurada**: Todas las estructuras de control tienen solo un punto de entrada y uno de salida

- ✓ C posee un repertorio de estructuras de control que permiten crear programas estructurados, aunque algunos detalles pueden violar la programación estruturada

## BLOQUES: SENTENCIAS COMPUESTAS

✓Es un conjunto de sentencia que se encierran entre los símbolos "{" y "}" para formar un bloque de código.

✓Pueden aparecer en cualquier sitio en el que podría aparecer una sentencia simple.

✓Pueden contener sentencias de declaración, pero solamento al comienzo del bloque (justo a continuación de '{'). El alcance de la declaración es el propio bloque.

✓La sintaxis es:

```
{
  sentencia;
        . . .
 sentencia;
}
```

## CONDICIONES EN C

✓ C no tiene el tipo booleano; para escribir condiciones:

- Una expresión que devuelva el valor 0 se considera valor 'Falso' y una expresión que devuelva un valor distinto de 0 se considera valor 'Verdadero'
- Se pueden escribir expresiones relacionales (***edad <3***)
- Se tienen operadores booleanos(***edad<18 && edad>0***)

## SENTENCIA ALTERNATIVA SIMPLE Y DOBLE if-else

| PSEUDOCÓDIGO | LENGUAJE C |
|---|---|
| **si** <condición><br>**entonces**<br>conjunto de sentencias<br>**si_no**<br>conjunto de sentencias<br>**fin_si** | **if (**<condición>**)**<br>**{**<br>conjunto de sentencias<br>**}**<br>**else**<br>**{**<br>conjunto de sentencias<br>**}** |

✓ Si el conjunto de sentencias está formado por una sola sentencia, pueden omitirse '{' y '}'

✓Puede omitirse la cláusula 'else' (alternativa simple).

✓ Un *if* anidado es un *if* que es el objeto de otro *if* o *else*. Una cláusula *else* siempre se refiere al *if* más próximo que esté en el mismo bloque que el *else* y que no esté asociado con un *if*.

Por ejemplo:

```
if(i) {
        if(j) sentencia 1;
        if(k) sentencia 2; /* este segundo if está */
        else sentencia 3; /* asociado con este else */
       }
```

Para asociar el 'else' el primer if, hay que delitar con '{' y '}' el bloque de sentencias asociado al primer if

```
if(i) {
        if(j){              /* Ahora este primer if está*/
            sentencia 1;
            if(k) sentencia 2;
        }
        else sentencia 3; /* asociado con este else */
}
```

✓ <u>El operador ternario ?:</u> Permite crear alternativas dentro de una expresión Exp1 **?** Exp2 : Exp3

Se evalua Exp1, si es cierta se evalua Exp2 y se convierte en el valor de la expresión; si Exp1 es falsa, se evalua Exp3 cuyo valor será el valor de la expresión

```
descuento= factura > 1000 ? 0.5 :0.5-0.4
```

## SENTENCIA ALTERNATIVA MÚLTIPLE switch

| PSEUDOCÓDIGO | LENGUAJE C |
|---|---|
| **segun_sea** (variable) **hacer**<br>1:<br>sentencias<br><br>2:<br>sentencias<br><br><br>....<br>en_otro_caso:<br>sentencias<br>**fin_segun** | **switch** (variable)<br>{<br>**case** valor1:<br>sentencias<br>**break**;<br>**case** valor2:<br>sentencias<br>**break**;<br>....<br>**default**:<br>sentencias<br>} |

- ✓ La variable usada para decidir la alternativa solo puede ser de tipo 'char' o 'int'

- ✓ Implícitamente se utiliza el operador de comparación '==' para compara la variable con cada uno de los casos.

- ✓ No puede haber dos 'case' con el mismo valor.

- ✓ Si se omite 'break' al final de cada caso, se ejecutan las sentencias del siguiente(s) caso, hasta encontrar un 'break' o si no hay, hasta llegar al final de la sentencia ('}'). Esto permite definir un mismo conjunto de sentencias para varios case

## BUCLE while

| PSEUDOCÓDIGO | LENGUAJE C |
|---|---|
| **mientras** (condición) **hacer**<br>sentencias<br>**fin_mientras** | **while** (condición)<br>**{**<br>sentencias;<br>**}** |

✓**break**: La ejecución de la sentencia break dentro de un bucle, provoca que el bucle finaliza inmediatamente y el control sigue en la sentencia posterior al bucle.

✓**continue**: En lugar de forzar la terminación, *continue* fuerza una nueva iteración del bucle y salta cualquier código que exista entre 'break' y la condición.

## BUCLE do-while

| PSEUDOCÓDIGO | LENGUAJE C |
|---|---|
| **repetir**<br><br>sentencias<br>**hasta_que** (condición); | **do**<br>**{**<br>sentencias;<br>**} while** (condición); |

✓La sentencia 'do-while' no es equivalente a un repetir hasta_que, ya que la condición es la misma que si se utiliza un 'mientras'. El 'do-while' debe ser visto como un 'while' cuyo cuerpo se ejecuta siempre al menos una vez.

✓ break y continue

## BUCLE for

| PSEUDOCÓDIGO | LENGUAJE C |
|---|---|
| **desde** i<-Vi **hasta** Vf **hacer**<br><br><br>secuencia de sentencias<br>**fin_desde** | **for(**inicialización; condición; incremento**)**<br>**{**<br>secuencia de sentencias;<br>**}** |

- ✓ El orden en que se suceden las cosas es:
  (1) inicialización, (2) comprobación de condición,
  (3) secuencia de sentencias, (4) incremento,
  (5) comprobación de condición . . . repitiendose los pasos (3), (4) (5) hasta finalizar el bucle

- ✓ Uso típico

  - inicialización: sentencia de asignación a una variable contador

  - condición: expresión relacional con la variable contador

  - incremento: incremento de la variable contador:

```
for (j=1; j<100;j++){

    printf("El cuadrado de %d es %d\n", j,j*j);
}
```

✓Usos atípicos:

(1) El bucle puede controlarse por una o por varias variables;

(2) La condición puede ser múltiple

(3) Si la variable de control del bucle tiene un valor previo, podemos obviar la inicialización

```
i=2;j=3;
for ( ; (i<10)&&(j>10) ; i=i+2, j=j-2)
      printf("%d %d\n",i,j);
```

✓**break**: Si omitimos todo, tenemos un bucle infinito, del que solo podremos salir con un break

```
for ( ; ;)
{  .......
  if (n ==100) break;
    ......
}
```

✓**continue**: Para el bucle *for*, *continue* hace que se ejecuten las partes de prueba condicional y de incremento del bucle