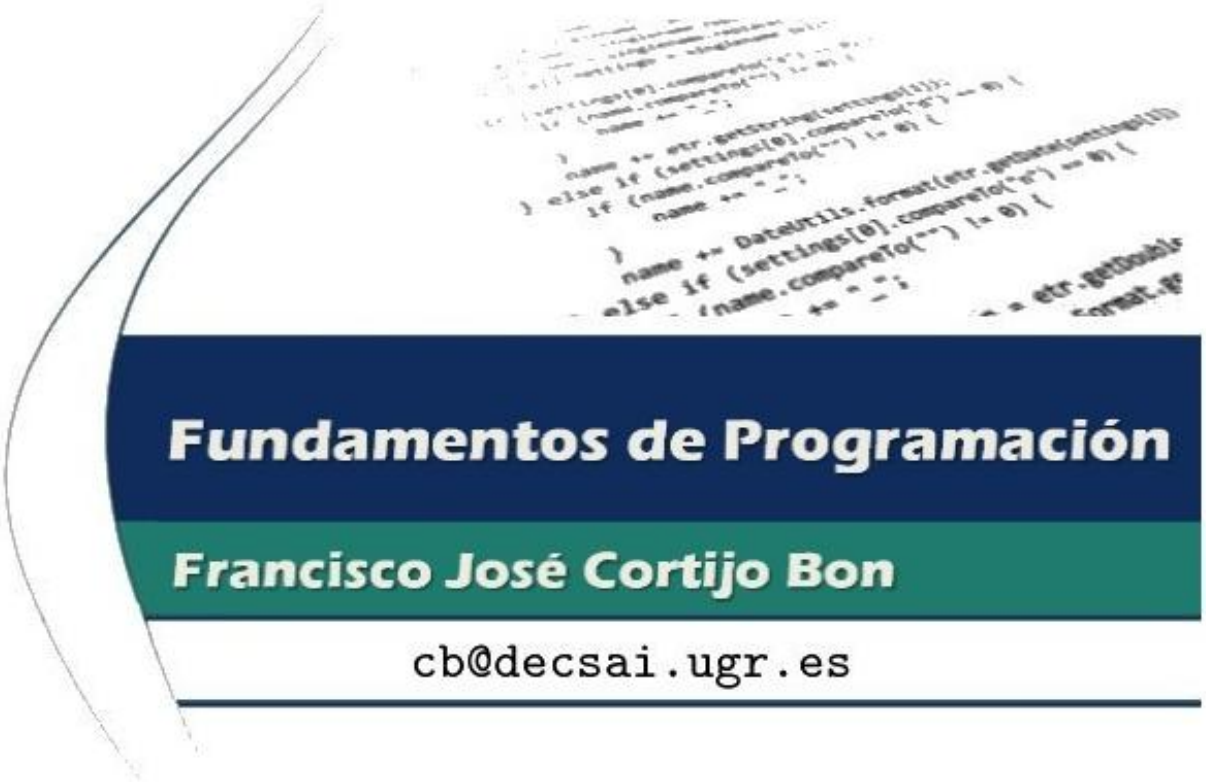
Departamento de Ciencias de la Computación e Inteligencia Artificial

Universidad de Granada



SEMINARIO SOBRE ESTRUCTURAS CONDICIONALES: UN PROBLEMA. DIFERENTES SOLUCIONES

Francisco José Cortijo Bon

Curso 2019/2020

1. Definición del problema

El problema a resolver es uno de los que aparecen en la Relación de Problemas II:

12. Ampliad el ejercicio 21 de la Relación de Problemas I. Después de haber leído los datos que definen el intervalo, el programa debe leer un valor real y determinar si está o no dentro del intervalo.

Finalidad: Plantear una condición compleja. Dificultad Media.

El ejercicio mencionado de la Relación de Problemas I dice:

21. Un intervalo es un espacio métrico comprendido entre dos valores o cotas, *a* y *b*, siendo *a* la cota inferior y *b* la cota superior. Cada extremo de un intervalo puede ser abierto o cerrado. En este problema no se consideran intervalos con extremos infinitos.

Construya un programa que lea e imprima los datos de un intervalo. Para ello, el programa debe leer los datos en el siguiente orden:
a) Un carácter que represente el tipo de intervalo por la izquierda: el usuario deberá introducir el carácter (si es abierto y [si es cerrado.
b) Un número real con la cota izquierda.
c) El carácter , como separador de las dos cotas
d) Un número real con la cota derecha.
e) Un carácter que represente el tipo de intervalo por la derecha: el usuario deberá introducir el carácter) si es abierto y] si es cerrado.

El programa mostrará en pantalla el mismo intervalo introducido.

Finalidad: Manejar entrada de datos de distinto tipo. Dificultad Baja.

2. Resolución del primer problema: lectura del intervalo.

En esta fase se leerán los caracteres y datos que determinan el intervalo en el formato indicado y se mostrará. Se comprobará la validez de los caracteres y valores introducidos (con los recursos conocidos hasta el momento).

En la solución propuesta se separan claramente las fases de entrada, cálculo y salida. Una vez leídos los datos se comprueba la validez de los mismos y después, sólo si todos los datos son correctos se procede a calcular el resultado y mostrarlo. Para poder separar las fases de cálculo y presentación de resultados, se emplea un **string** para componer el resultado.

Se propone la siguiente solución:

```
int main()
{
    const char ABIERTO_IZDA = '(';
    const char ABIERTO_DCHA = ')';
    const char CERRADO_IZDA = '[';
    const char CERRADO_DCHA = ']';
    const char SEPARADOR = ',';

    // Variables de entrada
    double a, b;                        // Cotas del intervalo
    char caracter_izda, caracter_dcha; // Delimitadores
    char separador;                     // Separador entre cotas


    // Entrada

    cout << "Escriba el intervalo: ";

    cin >> caracter_izda;
    cin >> a;
    cin >> separador;
    cin >> b;
    cin >> caracter_dcha;

    // Comprobar la validez de los caracteres y las cotas de entrada
    // y determinar el tipo de intervalo según los delimitadores.

    bool abierto_izda = false, abierto_dcha = false;
    bool cerrado_izda = false, cerrado_dcha = false;
    bool error_izda=false, error_dcha=false, error_separador=false;
    bool error_cotas = false;
    bool error;
```

Las variables bool declaradas actúan como *banderas* que se tomarán el valor **true** cuando se detecta el error al que se refieren. Por defecto, todas están fijadas a **false**. P.e. si el caracter separador no es **SEPARADOR** se modificará la variable **error_separador** para establecer su valor a **true**, indicando que hay un error en el separador.

```
    if (caracter_izda == ABIERTO_IZDA)
        abierto_izda = true;
    else
        if (caracter_izda == CERRADO_IZDA)
            cerrado_izda = true;
        else
            error_izda = true;

    if (caracter_dcha == ABIERTO_DCHA)
        abierto_dcha = true;
    else
        if (caracter_dcha == CERRADO_DCHA)
```

```
            cerrado_dcha = true;
        else
            error_dcha = true;
    if (separador != SEPARADOR) error_separador = true;

    if (a > b) error_cotas = true;
```

Una vez analizados los datos de engrada por separado pasamos a analizar la validez de los datos globalmente: en este caso, con que haya **alguno** incorrecto (basta uno sólo) deberemos terminar --> Usar el operador **||**

```
    // Cualquiera de los tres tipos de error impide continuar

    error = (error_izda||error_dcha||error_separador||error_cotas);

    if (!error) {

        // Separamos cálculos de la presentación.

        /*  Se trata de componer una dato string a partir de los
            datos leidos y darle la apariencia de un intervalo.
            Emplearemos el operador + para componer el string
            mediante concatenación y la función to_string para
            convertir los datos double a string.
        */

        // Cálculos (composición del string resultado)

        string intervalo; // Inicialmente, cadena vacía. Se podría
                          // inicializar explícitamente, pero no es
                          // preciso:    string intervalo = "";

        intervalo=intervalo+ caracter_izda+ to_string(a)+ separador;
        intervalo = intervalo + " " + to_string(b) + caracter_dcha;

         // Presentación del intervalo

        cout << endl;
        cout << "Intervalo: "  << intervalo;
        cout << endl;

    }
    else { // error

        cout << endl;
        cout << "Error en la entrada del intervalo." << endl;
        cout << endl;
    }

    return 0;
}
```

Esta solución satisface el requisito de no hacer ningún cálculo si los datos de entrada no son correctos.

3. Resolución del segundo problema.

La resolución del segundo problema se basa en la solución anterior: ya se ha leído el intervalo y se ha comprobado su validez. Por lo tanto, todo el código que se va a presentar está dentro del bloque:

```
if (!error) {
    ......

}
```

La solución consiste en evaluar la expresión que matemáticamente se plantea de manera sencilla pero que a la hora de programar debemos tener en cuenta:

1) hay cuatro tipos de intervalo: [a,b], [a,b), (a,b] y (a,b)

2) la comparación con los extremos a y b es lo que complica el algoritmo ya que el hecho de tener cuatro tipos de intervalo hace que la comprobación de los valores extremos dependa del tipo de intervalo. Así:

1) x $\in$[a,b] sii a <= x <=b
2) x $\in$[a,b) sii a <= x <b
3) x $\in$(a,b] sii a < x <=b
4) x $\in$(a,b) sii a < x <b

3.1 Versión "fuerza bruta"

```
    if (!error) {

        ......

        // Entrada: valor a comprobar

        double candidato;

        cout << "Valor a comprobar: ";
        cin >> candidato;


        // Calcular el tipo de intervalo
        //   [a,b] --> cerrado_cerrado
        //    [a,b) --> cerrado_abierto
        //    (a,b] --> abierto_cerrado
        //   (a,b) --> abierto_abierto
```

```
    bool cerrado_cerrado = (cerrado_izda && cerrado_dcha);
    bool cerrado_abierto = (cerrado_izda && abierto_dcha);
    bool abierto_cerrado = (abierto_izda && cerrado_dcha);
    bool abierto_abierto = (abierto_izda && abierto_dcha);


    bool pertenece;

    if (cerrado_cerrado) { // []

        if ((candidato >= a) && (candidato <= b))
             pertenece = true;
        else pertenece = false;
    }
    else {

        if (cerrado_abierto) { // [)

            if ((candidato >= a) && (candidato < b))
                 pertenece = true;
            else pertenece = false;
        }
        else {

            if (abierto_cerrado) { // (]

                if ((candidato > a) && (candidato <= b))
                     pertenece = true;
                else pertenece = false;
            }
            else {

                if (abierto_abierto) { // ()

                    if ((candidato > a) && (candidato < b))
                         pertenece = true;
                    else pertenece = false;
                }
            }
        }
    }

   // Salida

   if (pertenece) {
      cout << endl;
      cout << candidato << " pertenece a " << intervalo;
      cout << endl << endl;
   }
   else {
      cout << endl;
      cout << candidato << " NO pertenece a " << intervalo;
      cout << endl << endl;
   }

} // if (!error)
```

Observe cómo el uso de las variables **bool:**

```
bool cerrado_cerrado = (cerrado_izda && cerrado_dcha);
bool cerrado_abierto = (cerrado_izda && abierto_dcha);
bool abierto_cerrado = (abierto_izda && cerrado_dcha);
bool abierto_abierto = (abierto_izda && abierto_dcha);
```

simplifica las condiciones que se evalúan en los **if** y resultan más legibles.
Piense cómo se escribiría sin ellas las dos primeras condiciones, por ejemplo:

```
if (caracter_izda == CERRADO_IZDA &&
    caracter_dcha == CERRADO_DCHA) { // []

    if ((candidato >= a) && (candidato <= b))
        pertenece = true;
    else
        pertenece = false;
}
else {

    if (caracter_izda == CERRADO_IZDA &&
        caracter_dcha == ABIERTO_DCHA) { // ()
        ..................
```

3.2 Versión "resumida" (1)

Una vez se ha determinado el tipo de intervalo, cuando se evalúa la pertenencia o no, se asigna **true** o **false** a la variable **pertenece**. Para escribir menos código (e innecesario) podemos iniciar la variable **pertenece** a **false** (valor por defecto) y asignarle el valor **true** cuando detectemos la pertenencia al intervalo:

```
if (!error) {

    ......

    // Entrada: valor a comprobar

    double candidato;

    cout << "Valor a comprobar: ";
    cin >> candidato;


    // Calcular el tipo de intervalo
    //   [a,b] --> cerrado_cerrado
    //    [a,b) --> cerrado_abierto
    //    (a,b] --> abierto_cerrado
    //   (a,b) --> abierto_abierto
```

```
    bool cerrado_cerrado = (cerrado_izda && cerrado_dcha);
    bool cerrado_abierto = (cerrado_izda && abierto_dcha);
    bool abierto_cerrado = (abierto_izda && cerrado_dcha);
    bool abierto_abierto = (abierto_izda && abierto_dcha);


    bool pertenece = false;


    if (cerrado_cerrado) { // []

        if ((candidato >= a) && (candidato <= b))
            pertenece = true;
    }
    else {

        if (cerrado_abierto) { // [)

            if ((candidato >= a) && (candidato < b))
                pertenece = true;
        }
        else {

            if (abierto_cerrado) { // (]

                if ((candidato > a) && (candidato <= b))
                    pertenece = true;
            }
            else {

                if (abierto_abierto) { // ()

                    if ((candidato > a) && (candidato < b))
                        pertenece = true;
                }
            }
        }
    }

    // Salida

    if (pertenece) {
       cout << endl;
       cout << candidato << " pertenece a " << intervalo;
       cout << endl << endl;
    }
    else {
       cout << endl;
       cout << candidato << " NO pertenece a " << intervalo;
       cout << endl << endl;
    }

} // if (!error)
```

3.3 Versión "resumida" (2)

Si recuerda (ver apuntes) que una instrucción **if** cuyo *bloque if* contiene otra instrucción **if** equivale a una instrucción **if** con una condición compuesta con el AND de las dos anteriores, entonces podemos simplificar aún más el código.

```
if (C1)            equivale a   if (C1 && C2)
  if (C2)
```

```
if (!error) {

   ......

   // Entrada: valor a comprobar

   double candidato;

   cout << "Valor a comprobar: ";
   cin >> candidato;


   // Calcular el tipo de intervalo
   //   [a,b] --> cerrado_cerrado
   //    [a,b) --> cerrado_abierto
   //    (a,b] --> abierto_cerrado
   //   (a,b) --> abierto_abierto

   bool cerrado_cerrado = (cerrado_izda && cerrado_dcha);
   bool cerrado_abierto = (cerrado_izda && abierto_dcha);
   bool abierto_cerrado = (abierto_izda && cerrado_dcha);
   bool abierto_abierto = (abierto_izda && abierto_dcha);


   bool pertenece = false;


   if (cerrado_cerrado && ((candidato>=a)&&(candidato<= b)))
      pertenece = true;

   else
      if (cerrado_abierto && ((candidato >= a) && (candidato < b)))
         pertenece = true;

      else
         if (abierto_cerrado && ((candidato>a) && (candidato<=b)))
            pertenece = true;

         else
            if (abierto_abierto && ((candidato>a) && (candidato<b)))
               pertenece = true;
```

```
   // Salida

   if (pertenece) {
      cout << endl;
      cout << candidato << " pertenece a " << intervalo;
      cout << endl << endl;
   }
   else {
      cout << endl;
      cout << candidato << " NO pertenece a " << intervalo;
      cout << endl << endl;
   }

} // if (!error)
```

3.4 Versión "resumida" (3)

En la versión resumida (2) vemos claramente los cuatro casos en los que la variable **pertenece** debe ser **true**. ¿Por qué no asociarles variables lógicas a cada una de esas cuatro situaciones -casos- y asignar el valor a **pertenece** una sola vez?

```
if (!error) {

   ......

   // Entrada: valor a comprobar

   double candidato;

   cout << "Valor a comprobar: ";
   cin >> candidato;

   // Calcular el tipo de intervalo
   //   [a,b] --> cerrado_cerrado
   //    [a,b) --> cerrado_abierto
   //    (a,b] --> abierto_cerrado
   //   (a,b) --> abierto_abierto

   bool cerrado_cerrado = (cerrado_izda && cerrado_dcha);
   bool cerrado_abierto = (cerrado_izda && abierto_dcha);
   bool abierto_cerrado = (abierto_izda && cerrado_dcha);
   bool abierto_abierto = (abierto_izda && abierto_dcha);


   bool caso1=(cerrado_cerrado && ((candidato>=a)&&(candidato<=b)));
   bool caso2=(cerrado_abierto && ((candidato>=a)&&(candidato<b)));
   bool caso3=(abierto_cerrado && ((candidato>a) &&(candidato<=b)));
   bool caso4=(abierto_abierto && ((candidato>a) &&(candidato<b)));

   bool pertenece = caso1 || caso2 || caso3 || caso4;
```

```
   // Salida

   if (pertenece) {
      cout << endl;
      cout << candidato << " pertenece a " << intervalo;
      cout << endl << endl;
   }
   else {
      cout << endl;
      cout << candidato << " NO pertenece a " << intervalo;
      cout << endl << endl;

   }

} // if (!error)
```

3.5 Otra solución más compacta

Consideremos el caso de un intervalo abierto (a,b). Un valor x pertenece al intervalo si:

(x > a) && (x < b)

o sea, si el intervalo es abierto a la izquierda debemos asegurarnos que (x > a) mientras que si fuera cerrado a la izquierda comprobaríamos (x >= a). De la misma manera, si el intervalo es abierto a la derecha debemos asegurarnos que (x < b) mientras que si fuera cerrado a la derecha comprobaríamos (x <= b).

En resumen:

```
   ......
   bool valido_izda = ((candidato >= a) && (cerrado_izda)) ||
                      ((candidato > a) && (abierto_izda));
   bool valido_dcha = ((candidato <= b) && (cerrado_dcha)) ||
                      ((candidato < b) && (abierto_dcha));


   bool pertenece = valido_izda && valido_dcha;
   ......
```

Vale la pena pensar.