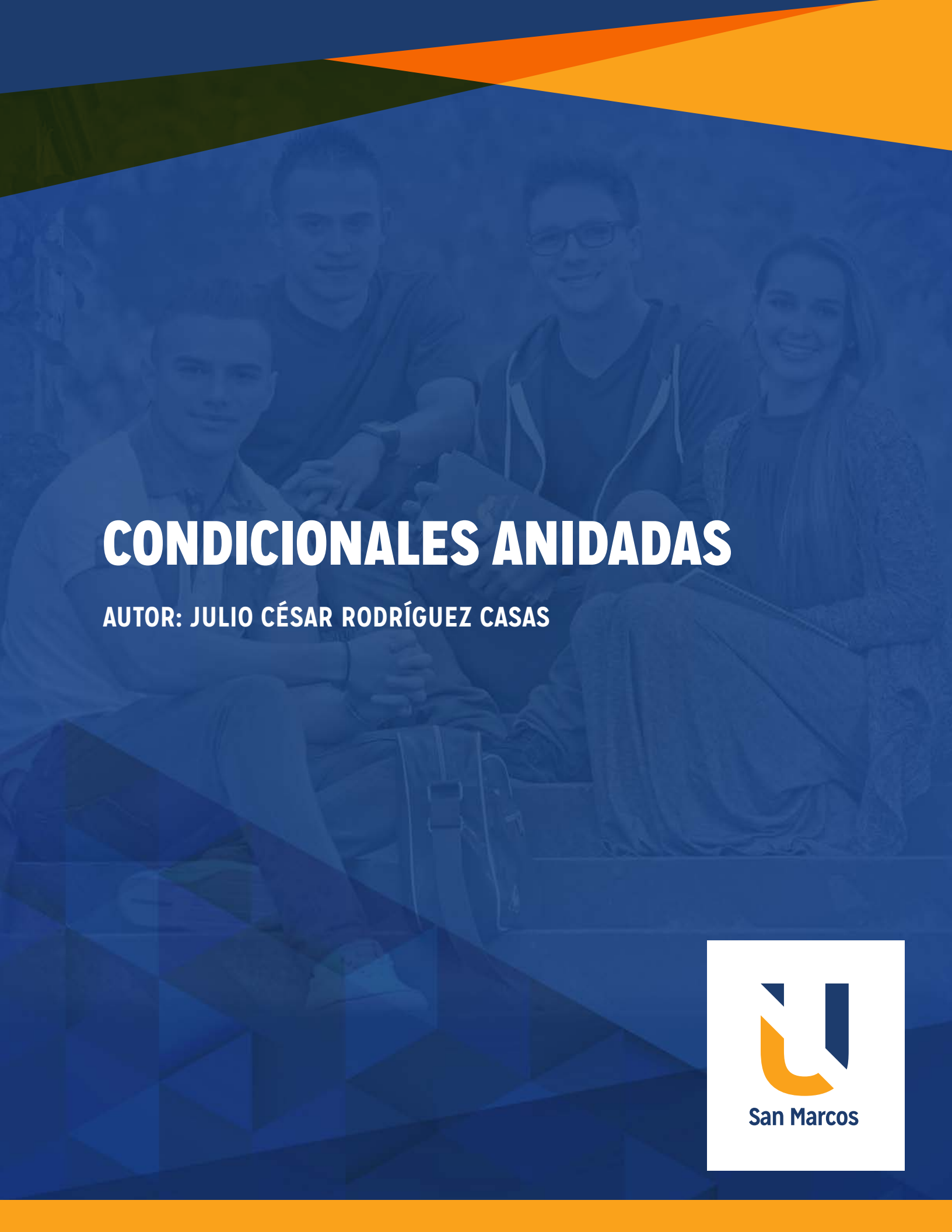


CONDICIONALES ANIDADAS

AUTOR: JULIO CÉSAR RODRÍGUEZ CASAS

San Marcos

Contenido

ÍNDICE

INTRODUCCIÓN

Introducción

El referente de pensamiento de este eje está basado en las estructuras de control. Veremos los tipos de estructuras como los saltos de línea que consisten en forzar el salto de punto del programa a otro que no va en orden secuencial. Pueden hacerse saliendo de un bloque de sentencias antes de acabarlo, o saltando de un punto a otro del programa mediante ciertas instrucciones comenzando por la condicional if-else, la cual nos permite tomar cierta decisión al interior de nuestro algoritmo, se trabajarán con algunos ejemplos con los cuales podrá practicar y familiarizarse con la estructura.

De igual forma, se verá la estructura switch case, que es más práctica cuando se tienen muchas condiciones que evaluar, se explicara su sintaxis y se realizarán ejemplos prácticos.

Seguidamente se analizará la estructura While y Do while y se explica con ejemplos el comportamiento de la estructura.

Por último, se mostrará el ciclo for, cada una de sus partes y ejemplos prácticos para ejecutar en el compilador de lenguaje C++.

A la par de este desarrollo temático se realizarán actividades de aprendizaje que permitirán poner en práctica estas estructuras.

Antes de continuar le recomendamos realizar la lectura complementaria acerca de las estructuras de control.

Lectura recomendada

Programación en C++/Iteraciones y decisiones.

bit.ly/28Pohla

Condicionales anidados

Condicionales anidados

Compilador de JDoodle
http://bit.ly/2gsosJx

Para la ejecución de los programas se puede utilizar los compiladores disponibles en línea, se recomienda utilizar el compilador de JDoodle.

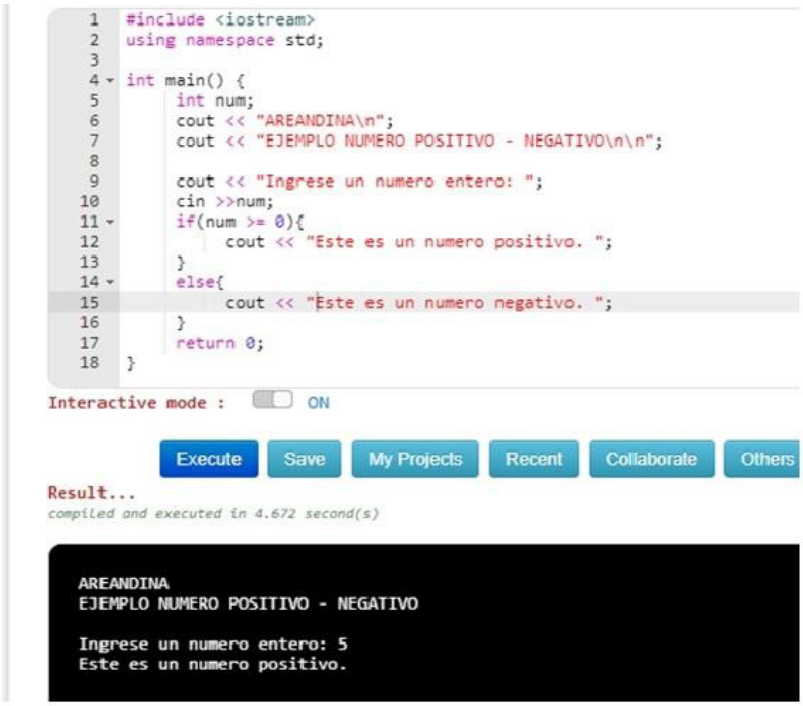


Figura 1. Compilador de Jdoodle
Fuente: Jdoodle

Se recomienda habilitar el modo interactivo (Interactive mode: ON), para interactuar con el programa; de igual forma, una vez se ejecuta el programa en la parte inferior se muestra el resultado de la compilación, bien sea errores de sintaxis o el resultado de la ejecución del programa.

Importante

Los programas presentados en el presente documento han sido escritos, aprobados y ejecutados en forma exitosa en el link anteriormente mencionado.

C++ usa todas las sentencias de control de ejecución de C. Esto incluye if-else, do-while, for, y una sentencia de selección llamada switch.

Sin embargo, lo primero que tenemos que hacer es agrupar varias sentencias en un bloque. para tratarlas todas como una unidad, a la que se le aplican unas determinadas órdenes. Para ello utilizaremos las llaves { ... } .

Dentro de las llaves escribiremos todas las sentencias que queramos agrupar, por ejemplo:

```
{
int a;
cout << "Tu edad : "; cin >> a;
cout << "tienes " << a << " años." << endl;
}
```

Estas sentencias forman un bloque al estar escritas dentro de las llaves.

La propia función int main () { ... } es un bloque, ya que todas las sentencias del programa deben escribirse dentro de las llaves.

Importante

La importancia de los bloques es fundamental para crear todo tipo de estructuras en donde queremos que afecten a más de una sentencia. Un bloque actúa de manera uniforme ante órdenes que se le den al mismo, como si fuera una única sentencia.

Tipos de estructuras

Una estructura cambia el flujo normal de ejecución del programa, de manera que este no se realice de una manera secuencial. Distinguimos entre las estructuras de control y otro tipo de estructuras como pueden ser las funciones o las clases.

Hay dos tipos de estructuras de control, que son las condicionales y los bucles. A esto se podría añadir un tercer tipo consistente en los saltos de línea, aunque este último es menos utilizado.

- Las condicionales consisten en que, llegado un punto, el programa deja de ser secuencial, y se bifurca, para elegir una opción entre dos (o entre varias) ignorando el resto, o pudiendo volver a ellas más tarde.

- Los bucles consisten en la repetición de determinadas tareas, La repetición no es exacta, en cada repetición hay alguna variable o algún elemento que cambia de manera controlada, así como una condición necesaria para que vuelva a repetirse la tarea. Cuando esta condición deja de existir la repetición se acaba y se sale del bucle.

Los saltos de línea consisten en forzar el salto de punto del programa a otro que no va en orden secuencial. Pueden hacerse saliendo de un bloque de sentencias antes de acabarlo, o saltando de un punto a otro del programa mediante ciertas instrucciones.

La estructura if

Dentro de las estructuras de control condicionales, la más importante es la estructura if. Esta consiste en que el programador indica una condición. Esta condición es evaluada como un dato booleano. Si el valor del dato es verdadero la condición se cumple, y, por lo tanto, la sentencia, o el bloque que va asociado a la condición, se ejecuta. Si el resultado de la condición es falso, la sentencia o bloque no se ejecuta.

Es decir, si la condición que hemos puesto se cumple, el programa realiza la acción, y si no es así no la realiza.

La sintaxis que se usa para esta estructura es la siguiente:

```
if (<condicion>) { <bloque de sentencias> }
```

Vamos cada uno de los componentes de esta estructura:

- if: es la palabra clave que indica el tipo de estructura. Esta palabra se pone siempre al principio.

- (<condicion>): entre paréntesis escribiremos a continuación una condición. Esta

tiene que dar un resultado booleano de “true” o “false”, por lo que en la mayoría de las veces suele ser una operación lógica o condicional, por ejemplo: (‘a’ > ‘b’).

- { <bloque de sentencias> }: después de la condición escribiremos el código al que queremos que afecte la estructura. este, si tiene más de una sentencia, debe ir en un bloque, (escrito entre llaves), ya que de otra manera solo afectaría a la primera sentencia.

Video

Se invita al estudiante a que vea el video acerca de cómo se realiza un algoritmo, en la página principal del eje.

Veamos un ejemplo de una estructura condicional con if:

```
#include<iostream>
using namespace std;
int main(){
    int a;
    cout << “Escribe un número mayor que 10 : “;
    cin >> a;
    if (a > 10) {
      cout << “Muy bien !! “ << endl;
      cout << “El numero “ << a << “ es mayor que 10.” << endl;
    }
}
```

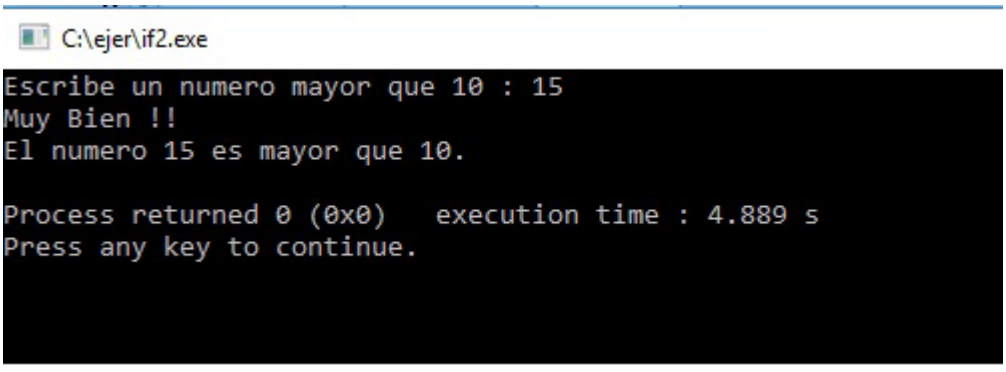


Figura 2. Resultado de la estructura condicional con if
Fuente: propia

Compilamos y ejecutamos el programa. Aquí pedimos al usuario que escriba un número mayor que 10. En la estructura condicional comprobamos que el número que nos ha dado el usuario sea mayor que 10, y solo en ese caso se ejecutará el código del bloque que está escrito entre llaves.

Observamos cómo se forma la estructura condicional. En primer lugar, la palabra clave if. Después de la condición entre paréntesis: (a > 10), y después la sentencia o bloque de sentencias.

```
    { cout << "Bien Hecho !! " << endl; cout << "El numero " << a << " es mayor que 10."
<< endl; }.
```

Aquí hemos puesto la estructura escrita en varias líneas, separando la condición, y tabulando las sentencias del bloque. Esto, no es imprescindible, ya que podríamos ponerlo todo seguido sin saltos de línea ni tabulaciones. por ejemplo:

```
  if (a > 10) { cout << "Muy Bien !! " << endl; cout << "El numero " << a << " es mayor que
10." << endl; }
```

Pero para una mayor claridad de los programadores, se suele poner siempre de la forma que lo hemos puesto antes:

```
  if (a > 10) {
   cout << "Muy Bien !! " << endl;
   cout << "El numero " << a << " es mayor que 10." << endl;
}
```

La palabra clave else

Si lo que queremos es que cuando la condición se cumpla se ejecute un código, y cuando no se cumpla se ejecute otro distinto, lo que debemos de hacer es añadir después de la estructura if la palabra clave else, seguida del bloque de sentencias que debe ejecutarse cuando la condición no se cumple. La estructura completa es:

```
  if (<condicion>) {
  <sentencias cuando SI se cumple>;
  }
else {
  <sentencias cuando NO se cumple>;
  }
```

Es decir, el programa evalúa la condición escrita después del if, y si el resultado es ver-

dadero se ejecuta el primer bloque de sentencias, pero si es falso se ejecutará el bloque que hay detrás del else.

Vamos a hacer lo mismo, pero con una estructura if ... else. El programa quedará así:

```
#include<iostream>

using namespace std;

int main(){

    int a;

    cout << "Areandina\n";

    cout << "Escribe un número mayor que 10 : ";

    cin >> a;

    if (a > 10) {

      cout << "Muy Bien  !! " << endl;

      cout << "El número " << a << " es mayor que 10." << endl;

    }

    else {

      cout << "Numero equivocado!! " << endl;

      cout << "El número " << a << " NO es mayor que 10." << endl;

    }

}
```

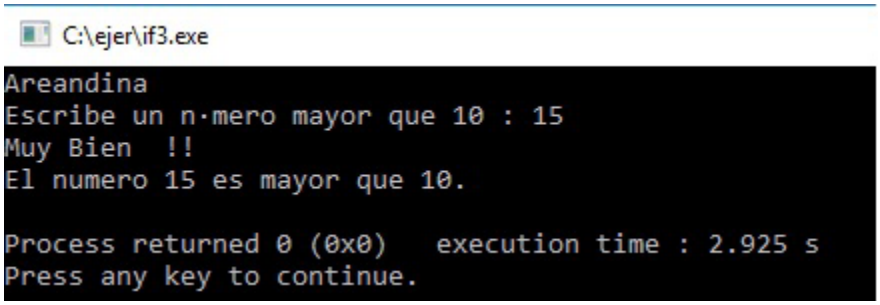


Figura 3. Resultado del ejemplo estructura if...else
Fuente: propia

Cuando el programa es ejecutado y evaluado por la parte falsa el resultado es el siguiente:

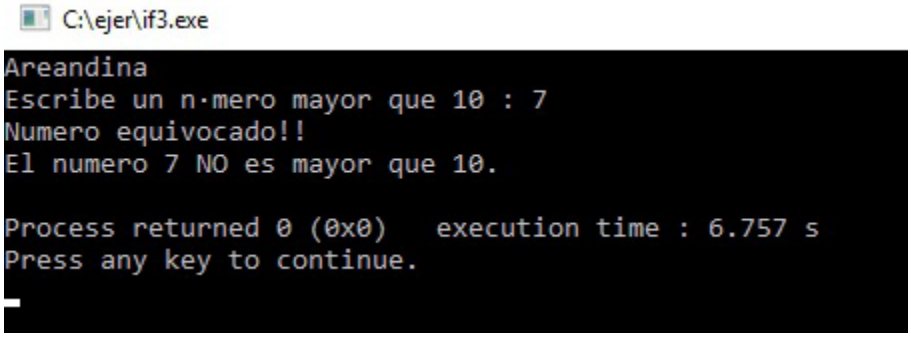


Figura 4. Resultado del ejemplo estructura if...else ejecutado
Fuente: propia

En este ejemplo pedimos dos números al usuario, el cual elige después la operación que va a realizar con ellos. Una vez elegida se realiza las operaciones aritméticas con esos dos números.

Veamos el código fuente:

```
#include<iostream>

using namespace std;

int main() {

    cout << "AREANDINA\n";

    cout << "OPERACIONES ARITMÉTICAS\n";

    cout << "OPERACIONES CON DOS NUMEROS" << endl;

    int a, b; char op;//Declaramos las variables

    cout << "Digite un numero: ";cin >> a;//pedimos el primer número

    cout << "Digite otro número: "; cin >> b;//pedimos el segundo número

    cout << "Los números son " << a << " y " << b << endl;//confirmamos números

     cout << "Elige una operación, para ello escribe su letra: " << endl;//explicación al
usuario

    cout << "S = Suma; R = Resta; M = Multiplicación; D = División." << endl;
```

```
    cout << "Tu operación : "; cin >> op;//pedimos operación
    if ((op == 's') || (op =='S')) {//opción 1: Suma.
        cout << "Operación: Suma " << endl;
        cout << a << " + " << b << " = " << a+b << endl;
    }
    else if ((op == 'r') || (op =='R')) {//opción 2: Resta
        cout << "Operación: Resta " << endl;
        cout << a << " - " << b << " = " << a-b << endl;
    }
    else if ((op == 'm') || (op =='M')) {//opción 3: Multiplicación
        cout << "Operación: Multiplicación " << endl;
        cout << a << " * " << b << " = " << a*b << endl;
    }
    else if ((op == 'd') || (op =='D')) {//opción 4 División
        cout << "Operación: División " << endl;
        cout << a << " / " << b << " = " << a/b << endl;
    }
    else {//Respuesta para opción equivocada.
        cout << "La letra escrita no se corresponde con ninguna operación." << endl;
    }
}
```

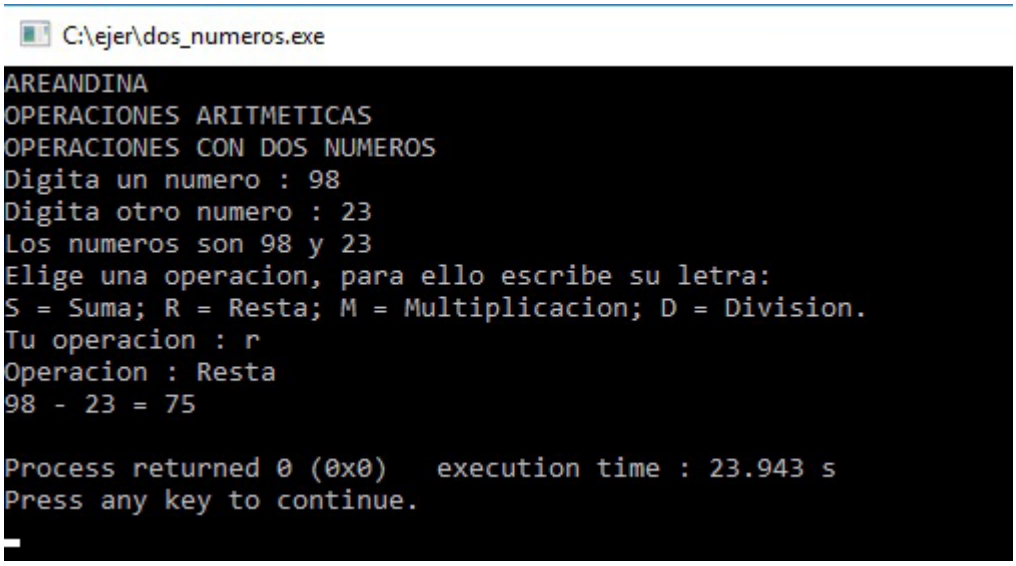


Figura 5. Ejemplo de resultado con operaciones aritméticas.
Fuente: propia

Video

Se invita al estudiante a que vea el video **cómo se utiliza el switch case** en la página principal del eje.

Condicional switch

La estructura switch tiene también la misma función if, pero aquí todas las opciones dependen del valor de una variable.

Importante

Tenemos una variable, y cada opción consiste en comparar esta variable con otro elemento, de manera que, si el valor de la variable y el del elemento comparado son iguales, el resultado es verdadero, la condición se cumple, y se ejecuta el código asociado. Si no es así se pasa a la siguiente opción donde se vuelve a repetir el proceso. Si no se ejecuta ninguna opción puede ponerse una opción por defecto al final.

Video

Antes de continuar veamos el video con un ejemplo sencillo sobre la **sentencia switch.**

youtu.be/InDTfd4XIX8

Código fuente

El código de esta estructura es el siguiente:

```
    switch (x) {
  case a:
   <sentencias para x == a>;
   break;
  case b:
   <sentencias para x == b>;
   break;
   .....................
  case n:
   <sentencias para x == n>;
   break;
  default:
   <sentencias para x == n>;
}
```

Vamos a explicar este código y ver cómo está estructurado:

- Empezamos poniendo la palabra clave switch, y después, entre paréntesis la variable con la que compararemos las demás (x).

- El resto del código forma un bloque y va escrito entre llaves. Este código se compone de varias opciones.

- Cada una de las opciones (excepto la última) empieza por la palabra clave case seguida del elemento con el que hacemos una comparación de igualdad y después dos puntos case a.

- Cada una de las opciones (excepto la última) acaba con la palabra clave break.

- En cada una de las opciones anteriores, entre el código del principio (case a:) y el del final (break;), escribiremos las sentencias que deben ejecutarse cuando la condición se cumpla.

- La opción final indica lo que debe ejecutarse en el caso en que ninguna de las anteriores se cumpla, y empieza por la palabra clave default: (acabada con dos puntos), seguida de las sentencias que deben ejecutarse cuando no se cumpla ninguna opción anterior.

Importante

Observa que podemos poner tantas opciones case .... break; como queramos. El case a: indica que iniciamos una nueva opción, la cual ejecutará el código que le sigue en el caso de que la variable principal (x) sea igual a la indicada a.

El break hace que el flujo se interrumpa y salgamos del bloque. De no ponerlo el programa no sale del bloque y el resto del código del bloque, perteneciente a otras opciones, también se ejecutaría.

La última opción default indica lo que se debe hacer cuando no hay ningún elemento anterior que coincida. Aunque no es obligatorio ponerla, es conveniente si queremos que el programa haga algo cuando no se cumplen las opciones anteriores. Al ser la última no es necesario poner el break; al final, ya que es ahí donde acabamos el bloque.

```
#include<iostream>

using namespace std;

int main() {

    cout << "AREANDINA\n";

    cout << "PROGRAMA SWITCH\n";

    cout << "OPERACIONES CON DOS NUMEROS" << endl;

    int a, b; char op;//Declaramos las variables

    cout << "Escribe un número : ";cin >> a;//pedimos el primer número

    cout << "Escribe otro número : "; cin >> b;//pedimos el segundo número

    cout << "Tus números son " << a << " y " << b << endl;//confirmamos números

    cout << "Elige una operación, para ello escribe su letra: " << endl;//explicación al
usuario

    cout << "s = Suma; r = Resta; m = Multiplicación; d = División." << endl;

    cout << "Tu operación: "; cin >> op;//pedimos operación

    switch (op){//inicio estructura.

      case 's': case 'S': case '+'://opcion 1: suma

        cout << "Operación : Suma " << endl;

        cout << a << " + " << b << " = " << a+b << endl;

        break;

      case 'r': case 'R': case '-'://opción 2: resta

        cout << "Operación: Resta " << endl;
```

```
            cout << a << " - " << b << " = " << a-b << endl;
            break;
        case 'm': case 'M': case '*'://opción 3: multiplicación
            cout << "Operación: Multiplicación " << endl;
            cout << a << " * " << b << " = " << a*b << endl;
            break;
        case 'd': case 'D': case '/'://opción 4 : división
            cout << "Operación: División " << endl;
            if (b != 0){
            cout << a << " / " << b << " = " << a/b << endl;
            }
            else {
              cout << "NO DIVIDIRÁS ENTRE CERO\n";
            }
            break;

        default: //Respuesta para opción equivocada.
            cout << "El carácter no se corresponde a ninguna operación." << endl;
    }
}
```

El siguiente es el resultado de la ejecución:

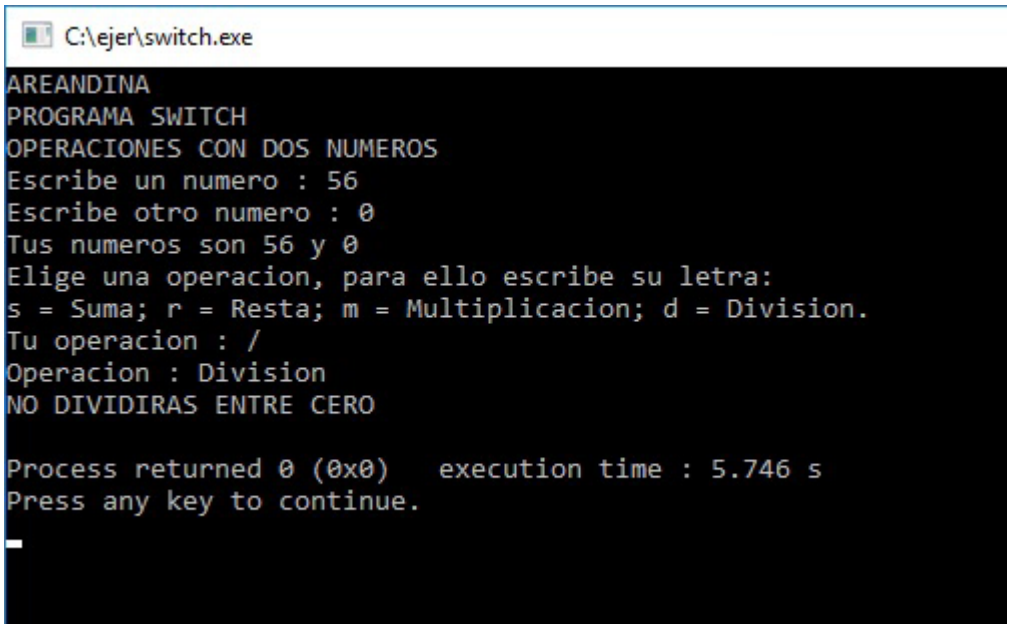


Figura 6. Resultado condicional switch
Fuente: propia

El bucle while

El tipo de bucle más sencillo es el bucle while. Para crear un bucle while tan solo se debe poner la palabra clave while seguida de la condición entre paréntesis, y después el bloque de sentencias.

```
while (<condicion>) {<Sentencias que se repiten>}
```

Sin embargo, la mayoría de las veces utilizaremos una operación condicional o lógica como condición. Por ejemplo, queremos que un determinado bloque se repita 10 veces. Para ello lo normal es utilizar una **variable de control**. Esta variable controlará el número de veces que se repite el bucle. Debemos declararla antes de empezar el bucle, y además inicializarla, normalmente con valor 0.

```
int i = 0;
```

Importante

Se utiliza una variable de tipo entero para controlar el número de vueltas (repeticiones) del bucle. Por convención suele utilizarse la letra i para definir esta variable.

Empezamos el bucle escribiendo la palabra clave while seguida de la condición entre paréntesis. Como queremos que el bucle se repita 10 veces, la condición será que la variable i se menor que 10:

```
while (i < 10)
```

Seguimos con el bloque de sentencias. Para que el bucle tenga un número definido de repeticiones, debemos variar la variable de control (i) de manera que cuente el número de repeticiones que llevamos en cada vuelta. Para ello, dentro del bloque de sentencias, debemos incrementar el valor de la variable de control una unidad en cada vuelta. Esto es lo que se llama "actualizar" la variable de control. El bloque de sentencias será como el siguiente:

```
{
<sentencias que se repiten>;
i = i+1;
}
```

Es decir, en cada vuelta el valor de i aumenta en una unidad. Cuando el valor de i sea igual a 10 la condición será falsa, y el bucle dejará de repetirse.

Veamos un ejemplo de programa utilizando while:

```
#include<iostream>
using namespace std;
int main() {
    cout << "AREANDINA\n";
    cout << "TABLA DE MULTIPLICAR\n";
    cout << "Tabla del nueve" << endl << endl;
    int i= 0;
    int tb = 9;
    while (i <=10) {
        cout << tb << " x " << i << " = " << tb * i << endl;
        i = i+1;
    }
}
```

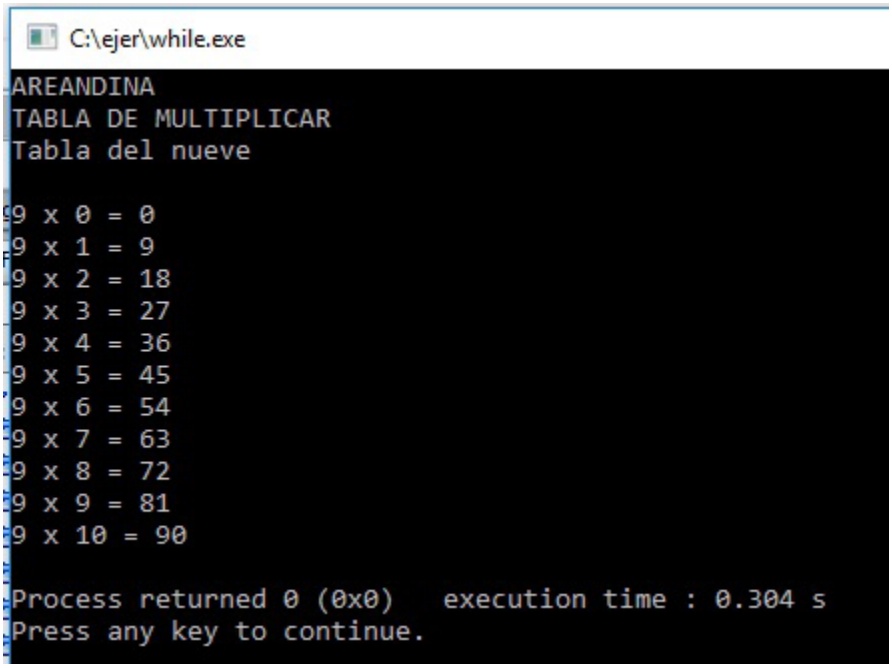


Figura 7. Ejemplo de programa utilizando while
Fuente: propia

Bucle do while

El bucle do while es muy parecido al bucle while. La diferencia es que aquí, aunque la condición no se cumpla, el código se ejecutará siempre al menos una vez.

Esto es debido a que primero se ejecuta el código, y después se comprueba la condición. Si esta es verdadera, el bucle se repite, si no es así se sale de él y continúa con el resto del programa.

Video

Antes de continuar, veamos en la página principal del eje, el video sobre **do while, cómo funciona, su sintaxis y un ejemplo básico.**

youtu.be/Z0kD_blq-Q4

Sintaxis de do while

La sintaxis es ligeramente diferente del bucle while, ya que aquí la condición se indica al final de la estructura:

```
   do {
  <Sentencias que se repiten>;
} while (<condicion>);
```

Empezamos poniendo la palabra clave do seguido del bloque de sentencias que se repiten. Después del bloque de sentencias ponemos la palabra clave while seguida de la condición entre paréntesis.

Tal como hemos indicado, al estar la condición al final del bucle, este se ejecuta siempre la primera vez, y la comprobación de la condición se hace al finalizar el bucle, con lo cual se decide si el bucle se repite o no.

Por lo demás es igual que el bucle while, es decir, si queremos repetir el bucle un número determinado de veces, necesitamos una variable de control (i).

Como con cualquier otro bucle, hay que tener en cuenta que tras un número de repeticiones tiene que llegar un momento en que la condición no se cumpla, para poder salir del bucle, y no caer en el error de hacer un bucle infinito.

Al igual que con el bucle while la variable de control la inicializamos antes del bucle, la utilizaremos también como referencia en la condición, y la actualizaremos en cada repetición del bucle. Por ejemplo, para hacer un bucle que se repita 10 veces haremos:

```
   int i=0;
do {
  <sentencias que se repiten>;
  i++;
} while (i < 10)
```

Veamos un ejemplo en el que empleamos el bucle do while. Aquí le pedimos al usuario un número y el programa repetirá una frase el número de veces que el usuario nos diga, pero si el número es cero o negativo, escribiremos igualmente la frase una vez. El archivo fuente será el siguiente:

```
        #include<iostream>

        using namespace std;

        int main() {
```

```
            int i=0, r;

            cout << "AREANDINA\n";

            cout << "PROGRAMA do while\n";

            cout << "Escribe un número para repetir la frase.";

            cin >> r;

             cout << "La frase se escribirá al menos una vez aunque el número sea cero o
negativo." << endl;

            do {

               i++;

               cout << i << ".- ALGORITMOS Y PROGRAMACIÓN. AREANDINA" << endl;

            } while(i < r);

        }
```

```
C:\ejer\do_while.exe
AREANDINA
PROGRAMA do while
Escribe un numero para repetir la frase.6
La frase se escribira al menos una vez aunque el numero sea cero o negativo.
1.- ALGORITMOS Y PROGRAMACION. AREANDINA
2.- ALGORITMOS Y PROGRAMACION. AREANDINA
3.- ALGORITMOS Y PROGRAMACION. AREANDINA
4.- ALGORITMOS Y PROGRAMACION. AREANDINA
5.- ALGORITMOS Y PROGRAMACION. AREANDINA
6.- ALGORITMOS Y PROGRAMACION. AREANDINA

Process returned 0 (0x0)   execution time : 6.327 s
Press any key to continue.
```

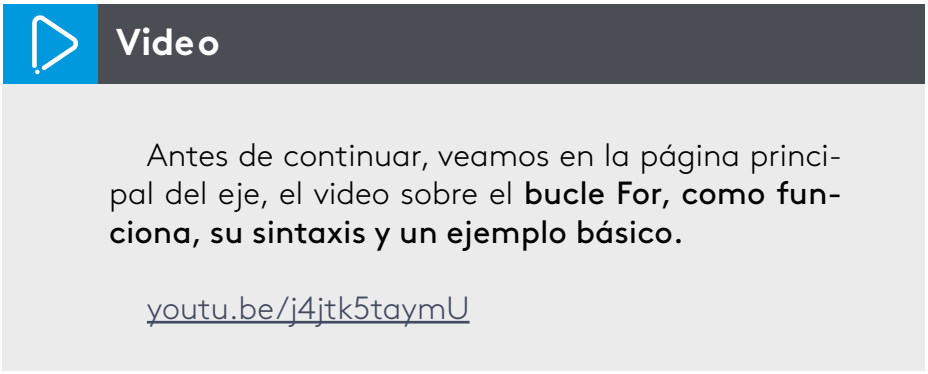
Figura 8. Ejemplo en el que empleamos el bucle do while
Fuente: propia

Bucle for

En la estructura for indicamos todo lo relacionado con la variable de control. Es decir, la estructura for no solo necesita incluir la condición, sino también el valor inicial de la variable de control (se llamará inicialización), y el cambio que se le aplica a la variable de control tras cada repetición (le llamaremos actualización).

En los bucles while tanto la inicialización como la actualización no forman parte de la misma estructura, y para ponerlos había que incluir la iniciación antes del bucle, y la actualización como una sentencia más del bloque que se repite.

En los bucles for tanto la inicialización como la actualización forman parte de la estructura, de manera que la variable de control es controlada en la misma estructura.



Sintaxis del bucle for

Entenderemos esto mejor viendo la sintaxis que utiliza este bucle. Esta es la siguiente:

```
for (<inicializacion>;<condicion>;<actualización>) {
 <sentencias que se repiten>;
}
```

Expliquemos el código de esta estructura:

- **for**: Empezamos el bucle escribiendo la palabra clave for.
- **<inicializacion>**: entre paréntesis y separados por punto y coma escribimos tres sentencias, la primera de ellas es la "inicialización", donde damos un valor inicial a la variable de control. Si esta no estaba declarada anteriormente, la podemos declarar aquí también ej.: int i=0;.
- **<condicion>**: la segunda sentencia del paréntesis es la "condición", en donde indicamos la condición que tiene que ser verdadera para que el bucle se repita. Evidentemente en la condición tiene que aparecer la variable de control. por ejemplo: i <= 10;.

- **<actualizacion>**: en esta tercera sentencia de dentro del paréntesis indicamos qué es lo que le debe ocurrir a la variable de control tras cada repetición, para que esta cambie y en un momento dado se pueda salir del bucle, en el ejemplo que estamos siguiendo, sería el incremento en una unidad: i++;.

- **{ <sentencias que se repiten>; }**: dentro de las llaves, escribimos el bloque de sentencias que se repetirán en cada vuelta.

Siguiendo con los ejemplos que hemos puesto antes podemos escribir un bucle que escriba una tabla de multiplicar, por ejemplo, la del 5:

```
for (int 1 = 0 ; i <= 10 ; i++) {
cout << "5 x " << i << " = " << 5*i << endl;
}
```

Dentro del paréntesis hemos puesto tres sentencias, la primera corresponde a la inicialización, en donde declaramos la variable i y la inicializamos a 0. En la segunda indicamos que esta variable debe ser menor o igual a 10 para que se repita el bucle. En la tercera indicamos que en cada vuelta se incrementará el valor de esta variable en una unidad.

En este programa vamos a pedirle al usuario que escriba un número entre el 1 y el 10 y el programa. El programa le devolverá la tabla de multiplicar de ese número. Si el número escrito no está entre el 1 y el 10, el programa le dirá al usuario que tiene un error y no puede escribir la tabla.

```
#include<iostream>

using namespace std;

int main() {

   cout << "AREANDINA\n";

   cout << "PROGRAMA BUCLE for\n";

  cout << "TABLAS DE MULTIPLICAR" << endl;

  int t;

  cout << "INTRODUZCA EL VALOR DE LA TABLA DE MULTIPLICAR (del 1 al 10) :";

  cin >> t;

  if (t >=1 and t<=10) {
```

```
    for (int i=1; i<=10 ; i++) {
      cout << t <<  " x " << i << " = " << t*i << endl;
    }
  }
  else {
    cout << "Error: tu número no está entre el 1 y el 10." << endl;
  }
}
```

Figura 9. Ejemplo de bucle que escriba una tabla de multiplicar
Fuente: propia

Lectura recomendada

Ahora bien, para complementar este tema, le invitamos a realizar la lectura complementaria sobre el bucle for.

http://bit.ly/2i2nzuB

Y para finalizar, les invitamos a realizar la actividad de repaso 1 para el desarrollo de algunos ejercicios que permiten profundizar en los temas vistos en el presente referente.

BIBLIOGRAFÍA

Bibliografía

Ceballos, F. (2007). Programación orientada a objetos con C++ (4a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11045993&p00=lenguaje+c%2B%2B

Ceballos, F. (2009). Enciclopedia del lenguaje C++ (2a. ed.). México: RA-MA Editorial. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046572&p00=lenguaje+c%2B%2B

Gaxiola, C., y Flores, D. (2008). Metodología de la programación con pseudocódigo enfocado al lenguaje C. México: Editorial Plaza y Valdés, S.A. de C.V. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10877093&p00=algoritmos+programaci%C3%B3n

Joyanes, L. (2006). Programación en C++: algoritmos, estructuras de datos y objetos (2a. ed.) Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491359&p00=lenguaje+c%2B%2B

Joyanes, L., Castillo, A., y Sánchez, L. (2005). C algoritmos, programación y estructuras de datos. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491350&p00=lenguaje+c%2B%2B

Joyanes, L., Rodríguez, L., y Fernández, M. (2003). Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos (2a. ed.) madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498607&p00=algoritmos+programaci%C3%B3n

Joyanes, L., y Sánchez, L. (2006). Programación en C++: un enfoque práctico. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491298&p00=lenguaje+c%2B%2B

Joyanes, L., y Zahonero, I. (2007). Estructura de datos en C++. Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10491301&p00=lenguaje+c%2B%2B

Joyanes, L. y Zahonero, I. (2005). Programación en C: metodología, algoritmos y estructura de datos (2a. ed.). Madrid: McGraw-Hill. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10498360&p00=algoritmos+programaci%C3%B3n

Juganaru, M. (2014). Introducción a la programación. México: Grupo Editorial Patria. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11017472&p00=programaci%C3%B3n

BIBLIOGRAFÍA

Moreno, J. (2014). Programación. México: RA-MA. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=11046398&p00=lenguaje+c%2B%2B

Noguera, F., y Riera, D. (2013). Programación. Barcelona: Editorial UOC. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10853420&p00=lenguaje+c%2B%2B

Schildt, H. (2009). C++: soluciones de programación. Madrid: McGraw-Hill Interamericana. Recuperado de http://site.ebrary.com.proxy.bidig.areandina.edu.co:2048/lib/bibliotecafuaasp/detail.action?docID=10433927&p00=algoritmos+programaci%C3%B3n

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica