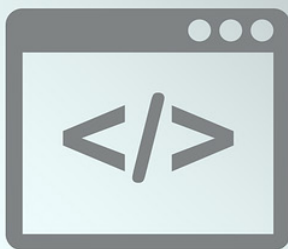


Fundamentos de **Programación** **Competitiva**



ECOE
EDICIONES

Hector Arturo Florez Fernandez

Fundamentos de programación competitiva

Hector Arturo Florez Fernandez

2024
Bogotá, Colombia

Flórez Fernández, Héctor Arturo, autor

Fundamentos de programación competitiva / Héctor Arturo Flórez Fernández. --
Primera edición. -- Bogotá : Ecoe Ediciones, 2024.

260 páginas, il., col.; 23.5x32 cm

(Computación y tecnología de la información. Desarrollo de software, Programación web, Lenguajes de programación y extensión/scripting: general).

Incluye datos curriculares de los autores -- Incluye referencias bibliográficas.

ISBN 978-958-508-285-4 (impreso) - 978-958-508-286-1 (pdf) -- 978-958-508-287-8 (digital)

1. Programación informática – concursos 2. Lenguajes de programación 3. C++ 4. Java
5. Python 6. Plataforma CodeForces 7. Algoritmos computacionales 8. Programación competitiva
– entrenamiento I. Flórez Fernández, Héctor Arturo, autor.

CDD: 005.115 Ed. 23



Área: Computación y tecnología de la información.

Subárea: Desarrollo de software - Programación web - Lenguajes de programación y extensión/scripting: general.

ECOE
EDICIONES



© Hector Arturo Florez Fernandez

Correo electrónico:

haflorezf@udistrital.edu.co

© Ecoe Ediciones S.A.S.

info@ecoeediciones.com

www.ecoeediciones.com

Carrera 19 # 63 C 32

Teléfono: (+57) 321 226 46 09

Bogotá, Colombia

♦ **Cita sugerida:**

Florez Fernandez, H.A. (2024).

Fundamentos de programación competitiva. Ecoe Ediciones.

Primera edición: Bogotá, Junio del 2024

ISBN: 978-958-508-285-4

e-ISBN (PDF): 978-958-508-286-1

e-ISBN (ePub): 978-958-508-287-8

Coordinadora editorial: Ana María Rueda G.

Coordinadora de producción editorial:

Alejandra Rondón Forero

Editora júnior de contenidos:

Alejandra Cely R.

Corrección de estilo: Hector Florez Fernandez

Carátula: Natalia Herrera Farfán

Impresión: Carvajal Soluciones de

Comunicación S.A.S.

Carrera 69 #15-24

*Prohibida la reproducción total o parcial por cualquier medio
sin la autorización escrita del titular de los derechos patrimoniales.*

Impreso y hecho en Colombia - Todos los derechos reservados

Sistema de Información en Línea



Al final del libro encontrará la información para ingresar al **Sistema de información en Línea - SIL** donde podrá acceder a:

- Código fuente en los lenguajes C++, Java y Python

Prólogo

El libro *Fundamentos de programación competitiva* ofrece al lector una exposición clara y suficiente de los conceptos básicos de programación que permitan adquirir las habilidades necesarias para el análisis y desarrollo de algoritmos.

El libro expone una gran cantidad de soluciones a problemas presentados en la plataforma CodeForces¹. Estas soluciones son desarrolladas en los lenguajes de programación C++, Java y Python. Los problemas estudiados en este libro se encuentran en inglés y no fueron traducidos al español con el fin de no correr el riesgo de generar interpretaciones incorrectas. En contraste, cada problema ha sido analizado para permitir un completo entendimiento y así poder llegar a una solución óptima. Estos problemas fueron cuidadosamente seleccionados entre un conjunto de mas de 8000 problemas disponibles en el *problemset*² de la plataforma Codeforces. Los problemas se escogieron con base en su facilidad de lectura, tema y nivel de dificultad.

Los capítulos que incluyen problemas tienen una sección denominada *Discusión* que tiene como objetivo explicar las decisiones tomadas en cada solución a cada problema además de hacer una comparación del desempeño de las soluciones en los lenguajes C++, Java y Python.

Este libro puede ser un gran apoyo para las personas con los siguientes perfiles:

- Profesor. El libro está diseñado para el aprendizaje de algoritmos, por lo tanto, puede apoyar a los profesores de cursos de algoritmia y programación.
- Estudiante. El libro está diseñado para que los estudiantes tengan un

¹<https://codeforces.com/>

²<https://codeforces.com/problemset>

aprendizaje autónomo que puede complementar los conocimientos y habilidades adquiridas en cursos de algoritmia y programación.

- **Entrenador.** La estructura del libro incluye varios temas que son de gran interés en los concursos de programación competitiva. El libro tiene muy en cuenta elementos que permiten mejorar el desempeño de las soluciones que son muy importantes para muchos tipos de problemas. De esta forma, un entrenador puede discutir con sus estudiantes las diferentes técnicas que permiten desarrollar algoritmos efectivos y eficientes.
- **Competidor.** El libro presenta una gran cantidad de problemas en diferentes temas relacionados con algoritmia y programación competitiva. Cada problema es detalladamente analizado para que el estudiante que desea competir en concursos de programación, comprenda completamente los requerimientos de cada problema y las soluciones propuestas en diferentes lenguajes de programación. De esta forma, cuando el estudiante compita en un concurso de programación competitiva, podrá tener los fundamentos para poder hacer un análisis crítico de nuevos problemas y así diseñar y desarrollar la mejor solución posible.

El autor del libro, Hector Arturo Florez Fernandez, es profesor titular de la Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia. Además, el contenido del libro fue revisado y discutido con varios miembros del semillero de Programación Competitiva³ y del grupo de investigación ITI: Innovación en Tecnologías de Información⁴ adscrito a la Universidad Distrital Francisco Jose de Caldas.

³<https://logiti.itiud.org/>

⁴<https://www.itiud.org/>

Tabla de contenido

Prólogo	i
Tabla de contenido	iii
Lista de figuras	vii
Lista de tablas	viii
Lista de algoritmos	ix
1 Lenguaje de programación C++	1
1.1 Conceptos básicos de programación usando el lenguaje C++ .	2
1.1.1 Operadores	2
1.1.2 Tipos Primitivos de Datos	3
1.1.3 Elementos básicos	4
1.1.4 Entrada y salida estándar	6
1.1.5 Función <code>main</code>	6
1.1.6 Estructuras de programación	6
2 Lenguaje de programación Java	14
2.1 Conceptos básicos de programación usando el lenguaje Java . .	16
2.1.1 Operadores	16
2.1.2 Tipos primitivos de datos	16
2.1.3 Elementos Básicos	17
2.1.4 Entrada y salida estándar	21
2.1.5 Método <code>main</code>	23
2.1.6 Estructuras de programación	24

3	Lenguaje de programación Python	32
3.1	Conceptos básicos de programación usando el lenguaje Python	33
3.1.1	Operadores	33
3.1.2	Tipos de datos	34
3.1.3	Elementos básicos	35
3.1.4	Entrada y salida estándar	36
3.1.5	Estructuras de programación	36
4	Complejidad algorítmica	41
4.1	Notación <i>Big-O</i>	41
4.2	Complejidad temporal	43
4.2.1	Complejidad constante	43
4.2.2	Complejidad lineal	43
4.2.3	Complejidad cuadrática	45
4.2.4	Complejidad cúbica	45
4.2.5	Complejidad logarítmica	46
4.2.6	Complejidad lineal logarítmica	48
4.2.7	Complejidad exponencial	49
4.2.8	Complejidad factorial	51
4.3	Complejidad espacial	52
4.3.1	Complejidad constante	52
4.3.2	Complejidad lineal	52
5	Cadenas de caracteres	53
5.1	Código ASCII	53
5.2	Operaciones frecuentes con cadenas de caracteres	58
5.3	Operaciones usando el código ASCII	64
5.4	Problemas	66
5.4.1	Way Too Long Words	66
5.4.2	Decoding	71
5.4.3	Chat room	76
5.4.4	cAPS LOCK	81
5.4.5	Palindromic Times	86
5.5	Discusión	91

6	Ordenamiento	93
6.1	Algoritmos de ordenamiento	93
6.1.1	Ordenamiento de burbuja	94
6.1.2	Ordenamiento por selección	95
6.1.3	Ordenamiento por inserción	98
6.1.4	Ordenamiento Shell	101
6.1.5	Ordenamiento por mezcla	103
6.1.6	Ordenamiento rápido	107
6.2	Funciones de ordenamiento	109
6.2.1	Función <code>sort</code> en C++	110
6.2.2	Función <code>stable_sort</code> en C++	112
6.2.3	Método <code>Arrays.sort</code> en Java	113
6.2.4	Función <code>sort</code> en Python	115
6.3	Problemas	117
6.3.1	Olympiad	117
6.3.2	Increasing	122
6.3.3	Towers	127
6.3.4	Promo	132
6.3.5	Social Distance	139
6.4	Discusión	149
7	Combinatoria	151
7.1	Combinación	152
7.1.1	Combinación sin repetición	152
7.1.2	Combinación con repetición	153
7.2	Permutación	154
7.2.1	Permutación sin repetición	154
7.2.2	Permutación con repetición	155
7.3	Variación	156
7.3.1	Variación sin repetición	156
7.3.2	Variación con repetición	157
7.4	Propiedades de números combinatorios	158
7.5	Teorema del binomio	159
7.6	Triángulo de Pascal	160
7.7	Problemas	162

7.7.1	Password	162
7.7.2	CQXYM Count Permutations	167
7.7.3	Infinite Replacement	173
7.7.4	Lucky Numbers	178
7.7.5	Silent Classroom	182
7.8	Discusión	191
8	Teoría de números	192
8.1	Divisibilidad	193
8.2	Números primos	194
8.2.1	Prueba de primalidad	194
8.2.2	Siguiente número primo	197
8.2.3	Criba de Eratóstenes	198
8.3	Teorema fundamental de la aritmética	201
8.4	Máximo común divisor	203
8.5	Mínimo común múltiplo	206
8.6	Problemas	208
8.6.1	Gregor and Cryptography	208
8.6.2	Factorise N+M	211
8.6.3	Almost Prime	217
8.6.4	I'm bored with life	223
8.6.5	Minimum LCM	227
8.7	Discusión	236
	Bibliografía	238

Lista de figuras

4.1	Gráfica de complejidad de <i>Big-O</i>	42
4.2	Árbol de recursividad de la complejidad exponencial	50
4.3	Árbol de recursividad de la complejidad factorial	52
6.1	Ordenamiento de burbuja	95
6.2	Ordenamiento por selección	97
6.3	Ordenamiento por inserción	99
6.4	Ordenamiento Shell	102
6.5	Ordenamiento por mezcla	105
6.6	Ordenamiento rápido	108
8.1	Criba de Eratóstenes. Creación de lista	199
8.2	Criba de Eratóstenes. Lista con marcación de múltiplos de 2 .	199
8.3	Criba de Eratóstenes. Lista con marcación de múltiplos de 2 y 3	199
8.4	Criba de Eratóstenes. Lista con marcación de múltiplos de 2, 3 y 5	200
8.5	Criba de Eratóstenes. Lista con marcación de múltiplos de 2, 3, 5 y 7	200

Lista de tablas

1.1	Operadores aritméticos en C++	2
1.2	Operadores de asignación en C++	2
1.3	Operadores lógicos en C++	3
1.4	Operadores de comparación en C++	3
1.5	Operadores a nivel de bits en C++	3
1.6	Tipos primitivos de datos en C++	4
2.1	Operadores aritméticos en Java	16
2.2	Operadores de asignación en Java	17
2.3	Operadores lógicos en Java	17
2.4	Operadores de comparación en Java	17
2.5	Operadores a nivel de bits en Java	18
2.6	Tipos primitivos de datos en Java	19
3.1	Operadores aritméticos en Python	33
3.2	Operadores de asignación en Python	34
3.3	Operadores lógicos en Python	34
3.4	Operadores de comparación en Python	34
5.1	Código ASCII	54

Lista de algoritmos

4.1	Función con orden de complejidad temporal constante	44
4.2	Función con orden de complejidad temporal lineal	44
4.3	Función con orden de complejidad temporal cuadrática	45
4.4	Función con orden de complejidad temporal cubica	46
4.5	Función iterativa con orden de complejidad temporal logarítmica	47
4.6	Función recursiva con orden de complejidad temporal logarítmica	48
4.7	Función con orden de complejidad temporal lineal logarítmica .	49
4.8	Función con orden de complejidad temporal exponencial	50
4.9	Función con orden de complejidad temporal factorial	51
6.1	Ordenamiento de burbuja	96
6.2	Ordenamiento por selección	98
6.3	Ordenamiento por inserción	100
6.4	Ordenamiento Shell	103
6.5	Ordenamiento por mezcla	106
6.6	Ordenamiento rápido	109
8.1	Prueba de primalidad versión 1	195
8.2	Prueba de primalidad versión 2	195
8.3	Prueba de primalidad versión 3	196
8.4	Prueba de primalidad versión 4	196
8.5	Prueba de primalidad versión 5	197
8.6	Siguiente número primo	198
8.7	Criba de Eratóstenes	201
8.8	Descomposición en factores primos	203
8.9	MCD - Algoritmo mediante iteraciones	204
8.10	MCD - Algoritmo de Euclides	205
8.11	MCD - Algoritmo por descomposición de factores primos	206

1

Lenguaje de programación C++

C++ fue desarrollado en los laboratorios Bell en 1980 por Bjarne Stroustrup. El objetivo principal de desarrollar C++ fue incluir el paradigma de programación orientada a objetos al lenguaje C agregando clases y objetos. Inicialmente este lenguaje fue llamado *C con clases*, pero en 1983 fue renombrado a C++ que significa *C incrementado* basándose en el operador ++ de C. C++ estuvo disponible al público en 1985 a pesar de que aun no se encontraba estandarizado. C++ fue actualizado en 1989 para incluir herencia, miembros protegidos y estáticos [14].

C++ es considerado un lenguaje de nivel medio debido a que combina características de bajo y alto nivel. C++ es aun un lenguaje muy reconocido por su portabilidad, buen desempeño y funcionalidad de bajo nivel. Este lenguaje incluye todas las características del lenguaje C además de clases, herencia, sobrecarga de métodos, tipado fuerte de datos, manejo de excepciones, entre otros.

En 1985, Bjarne Stroustrup publicó el texto *The C++ Programming Language*, el cual se convirtió en el manual de referencia del lenguaje C++. En 1990, Stroustrup publicó *The Annotated C++ Reference Manual*, el cual fue

base para el desarrollo del estándar C++ ISO/IEC 14882:1998 publicado en 1998.

1.1 Conceptos básicos de programación usando el lenguaje C++

1.1.1 Operadores

En C++ existen diferentes tipos de operadores que permiten realizar diferentes procesos. Estos operadores se clasifican en aritméticos, de asignación, lógicos, de comparación y a nivel de bits. Los operadores aritméticos se presentan en la Tabla 1.1, los operadores de asignación se presentan en la Tabla 1.2, los operadores lógicos se presentan en la Tabla 1.3, los operadores de comparación se presentan en la Tabla 1.4 y los operadores a nivel de bits se presentan en la Tabla 1.5.

Tabla 1.1: Operadores aritméticos en C++

Operador	Símbolo	Implementación
Suma	+	$a = b + c$
Resta	-	$a = b - c$
Multiplicación	*	$a = b * c$
División	/	$a = b / c$
Módulo	%	$a = b \% c$
Incremento	++	$a++$
Decremento	--	$a--$

Tabla 1.2: Operadores de asignación en C++

Operador	Símbolo	Implementación
Asignación	=	$a = b$
Asignación con suma	+=	$a += b \Leftrightarrow a = a + b$
Asignación con resta	-=	$a -= b \Leftrightarrow a = a - b$
Asignación con multiplicación	*=	$a *= b \Leftrightarrow a = a * b$
Asignación con división	/=	$a /= b \Leftrightarrow a = a / b$

Tabla 1.3: Operadores lógicos en C++

Operador	Símbolo	Implementación	Observación
<i>NOT</i>	!	!a	
<i>AND</i>	&&	a = b && c	Si b es falso, no evalúa c
<i>OR</i>		a = b c	Si b es verdadero, no evalúa c
<i>AND</i>	&	a = b & c	Siempre se evalúan a y b
<i>OR</i>		a b	Siempre se evalúan a y b

Tabla 1.4: Operadores de comparación en C++

Operador	Símbolo	Implementación
Igual	==	a == b
Diferente	!=	a != b
Mayor	>	a > b
Menor	<	a < b
Mayor o igual	>=	a >= b
Menor o igual	<=	a <= b

Tabla 1.5: Operadores a nivel de bits en C++

Símbolo	Implementación	Descripción
>>	a >> b	Desplaza a la derecha los bits de a una distancia b
<<	a << b	Desplaza a la izquierda los bits de a una distancia b
&	a & b	AND a nivel de bits
	a b	OR a nivel de bits
^	a ^ b	XOR a nivel de bits
~	~a	Complemento a nivel de bits

1.1.2 Tipos Primitivos de Datos

La Tabla 1.6 presenta los tipos primitivos de datos y sus características principales.

Tabla 1.6: Tipos primitivos de datos en C++

Tipo	Descripción	Tamaño	Rango de Valores
<i>byte</i>	Número entero	1 byte	-128 a 127
<i>short</i>	Número entero	2 bytes	-32768 a 32767
<i>int</i>	Número entero	4 bytes	-2.147.483.648 a 2.147.483.647
<i>long</i>	Número entero	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
<i>float</i>	Número flotante con precisión simple	4 bytes	-3.402823E38 a -1.401298E-45 y 1.401298E-45 a 3.402823E38
<i>double</i>	Número flotante con precisión doble	4 bytes	-1.79769313486232E308 a -4.94065645841247E-324 y 4.94065645841247E-324 a 1.79769313486232E308
<i>char</i>	Carácter Unicode	2 bytes	Símbolos Unicode
<i>bool</i>	Valor booleano	1 byte	true , false

1.1.3 Elementos básicos

Sentencia

En C++, una sentencia tiene al final el símbolo punto y coma (;). Es posible incluir varias sentencias en una línea de código, sin embargo, se considera una buena práctica utilizar una línea para cada sentencia. Las siguientes líneas son ejemplos de sentencias.

```

1 | int a;
2 | int b;
3 | bool f;
4 | c = a + b;
5 | f = (a >= 10) && (b <= 20);

```

Comentario

Existen dos formas diferentes de introducir comentarios en el código fuente de una aplicación en C++. La primera consiste en colocar el símbolo // en una línea de código y en seguida el texto del comentario. La segunda consiste en incluir el símbolo /* al inicio del comentario y el símbolo */ al final del

comentario. Esta segunda forma permite hacer comentarios en varias líneas de código. Las siguientes líneas son ejemplos de comentarios.

```
1 //Este es un comentario en una línea de código
2 /*
3 Este es un comentario
4 En diferentes líneas de código
5 */
```

Arreglo

En C++, un arreglo es una estructura de datos que permite almacenar un conjunto de valores de un mismo tipo de dato. El tamaño de un arreglo se debe indicar al momento de crear el arreglo y este tamaño no puede ser modificado. Los elementos de un arreglo se acceden mediante índices enteros, donde el primer índice es 0. Los arreglos pueden ser unidimensionales o multidimensionales. Los arreglos de dos dimensiones son conocidos como matrices.

Las siguientes líneas de código crean un arreglo de números enteros de 10 posiciones, un arreglo de cadena de caracteres de 10 posiciones y una matriz de números enteros de 3 filas y 3 columnas.

```
1 int a[10];
2 string b[10];
3 int c[3][3];
```

Un arreglo también puede crearse incluyendo valores iniciales. La siguiente línea de código crea un arreglo de tres números enteros.

```
1 int x[] = {10,20,30};
```

La longitud de un arreglo se calcula con la función `sizeof`, sin embargo, esta función retorna la cantidad de bytes que utiliza el arreglo. Por lo tanto, para conocer la cantidad de elementos de un arreglo, se calcula la cantidad de bytes del arreglo y se divide entre la cantidad de bytes de un elemento del arreglo, la cual se puede calcular con la misma función `sizeof`. La cantidad de bytes de un elemento del arreglo, también se puede calcular usando el tipo de dato del arreglo.

Las siguientes líneas de código crea un arreglo de tres números enteros y calcula la cantidad de elementos del arreglo mediante la función `sizeof` usando un elemento del arreglo y usando el tipo de dato del arreglo.

```
1 | int x[] = {10,20,30};  
2 | int n = sizeof(x)/sizeof(x[0]);  
3 | int m = sizeof(x)/sizeof(int);
```

1.1.4 Entrada y salida estándar

C++ tiene la función `cin` que permite ingresar datos mediante entrada estándar (que equivale a entrada por teclado mediante consola o terminal) una cadena de caracteres y la función `cout` que permite imprimir datos en salida estándar (que equivale a impresión en consola o terminal).

El siguiente código fuente permite capturar una cadena de caracteres desde entrada estándar e imprimirlo en salida estándar.

```
1 | string x;  
2 | cin >> x;  
3 | cout << x;
```

1.1.5 Función `main`

La función `main` permite iniciar la ejecución de una aplicación en C++. Esta función tiene como retorno el tipo de dato `int`.

El siguiente código fuente presenta la implementación de la función `main` con impresión del mensaje *Hola Mundo*.

```
1 | int main(){  
2 |     cout << "Hola mundo";  
3 | }
```

1.1.6 Estructuras de programación

Estructura de condición `if`

En C++, la estructura de condición `if` se compone de una condición, la cual debe encontrarse entre paréntesis. Tiene un conjunto de instrucciones que debe estar incluido entre los símbolos `{` y `}`. Sin embargo, si solo se desea ejecutar una instrucción no es necesario incluir los símbolos `{` y `}`.

El siguiente código fuente presenta un ejemplo que captura un número entero en la variable `x` y si el valor de la variable `x` es par, esta variable se duplica. Finalmente, se imprime el valor de la variable `x`.

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     int x;
5     cin >> x;
6     if(x%2 == 0){
7         x *= 2;
8     }
9     cout << x
10 }

```

Resultado	
Entrada estándar	Salida estándar
2	4
3	3

Estructura de condición **if else**

En C++, la estructura de condición **if else** se compone de una condición, la cual debe encontrarse entre paréntesis. Tiene dos conjuntos de instrucciones que deben estar incluidos entre los símbolos { y }.

El siguiente código fuente presenta un ejemplo en donde se captura un número entero en la variable **x** e imprime si el número es par o impar.

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     int x;
5     cin >> x;
6     if(x%2 == 0){
7         cout << x << " es par";
8     }else{
9         cout << x << " es impar";
10    }
11 }

```

Resultado	
Entrada estándar	Salida estándar
2	2 es par
3	3 es impar

Estructura de condición `if else if`

En C++, la estructura de condición `if else if` se compone de múltiples condiciones, las cuales deben encontrarse entre paréntesis. Todos los conjuntos de instrucciones deben estar incluidos entre los símbolos `{ y }`. La primer condición debe estar antecedida de la sentencia `if` y el resto de condiciones deben estar antecedidas de la sentencia `else if`. Esta estructura permite tener opcionalmente al final una estructura `else`.

El siguiente código fuente presenta un ejemplo en donde se captura un número entero en la variable `x` e imprime si el número es positivo, negativo o neutro.

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     int x;
5     cin >> x;
6     if(x < 0){
7         cout << x << " es negativo";
8     }else if(x > 0){
9         cout << x << " es positivo";
10    }else{
11        cout << x << " es neutro";
12    }
13 }
```

Resultado	
Entrada estándar	Salida estándar
-5	-5 es negativo
5	3 es positivo
0	0 es neutro

Operador ternario

En C++, La estructura de condición **if else** se puede implementar con una sintaxis que permite la codificación en una sola línea, la cual se denomina operador ternario. Este operador se debe aplicar si y solo si se cuenta con un solo proceso en el bloque **if** y en el bloque **else** y tienen el mismo comportamiento.

El siguiente código fuente presenta un ejemplo en donde se captura un número entero en la variable **x** e imprime si el número es par o impar.

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     int x;
5     cin >> x;
6     string paridad = (x%2 == 0)?"par":"impar";
7     cout << x << " es " << paridad;
8 }
```

Resultado	
Entrada estándar	Salida estándar
2	2 es par
3	3 es impar

Estructura de condición **switch case**

En C++, la estructura de condición **switch case** se compone de múltiples condiciones sobre una misma variable las cuales siempre deben arrojar un valor booleano. La variable sujeto de la condición debe encontrarse entre paréntesis posterior a la sentencia **switch**. Los valores de la variable deben estar posterior a la sentencia **case** seguida del carácter dos puntos (:).

Posterior a cada **case**, se implementan el conjunto de instrucciones del proceso correspondiente. Al finalizar las instrucciones, se debe colocar la sentencia **break** con el fin de salir del proceso. En caso de que no se coloque la sentencia **break** en un proceso, se ejecutará el proceso del siguiente **case** hasta que se encuentre la sentencia **break**. De esta forma, es posible ejecutar varios procesos incluidos en varios **case**. Opcionalmente, posterior al último **case**, se puede incluir un proceso mediante la sentencia **default**, el cual ejecutaría

el correspondiente conjunto de instrucciones si ninguno de los anteriores **case** es ejecutado.

Por otro lado, la variable de la sentencia **switch**, solo puede contener datos de tipo entero o caracter.

El siguiente código fuente presenta un ejemplo en donde se captura un número entero en la variable **x**, luego se captura un numero en la variable **s** y se ejecuta las siguientes condiciones:

- Si $s = 1 \rightarrow x = x * 2 + 10$
- Si $s = 2 \rightarrow x = x + 10$
- Si $s = 3 \rightarrow x = x - 10$
- Si $s \neq 1, s \neq 2, s \neq 3 \rightarrow x = 0$

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     int x;
5     int s;
6     cin >> x;
7     cin >> s;
8     switch(s){
9         case 1:
10             x *= 2;
11         case 2:
12             x += 10;
13             break;
14         case 3:
15             x -= 10;
16             break;
17         default:
18             x = 0;
19     }
20     cout << x;
21 }
```