Tema 2

Conceptos básicos de C++

En este capítulo vamos a introducir el lenguaje de programación C++ que se va a utilizar en la asignatura, describiendo algunos conceptos básicos y plasmando la estructura general de un programa escrito en este lenguaje. A lo largo de los sucesivos temas se seguirá la notación BNF para ir describiendo las diferentes reglas sintácticas del lenguaje. De esta forma se irá introduciendo de una manera formal la sintaxis de C++.

Índice General

2.1 Elementos del lenguaje

Comenzaremos viendo los elementos más simples que integran un programa escrito en C++, es decir palabras, símbolos y las reglas para su formación.

1. Identificadores

Son nombres elegidos por el programador para representar entidades (variables, tipos, constantes, procedimientos, módulos) en el programa. El usuario puede elegir cualquier identificador excepto un pequeño grupo que se reserva para usos específicos.

C++ distingue entre letras mayúsculas y minúsculas, por lo que a efectos del programa serán identificadores distintos `hola`, `Hola` o `hoLa`.

No se impone ningún límite en cuanto a la longitud de los identificadores.

```
<Ident> ::= <letra> {<letra>|<digito>}
```

2. Palabras reservadas

   Tienen un significado predeterminado para el compilador y sólo pueden ser usadas con dicho sentido.

and	and_eq	asm	auto	bitand	bitor
bool	break	case	catch	chat	class
compl	const	const_cast	continue	default	delete
do	double	dynamic_cast	else	enum	explicit
export	extern	false	float	for	friend
goto	if	inline	int	long	mutable
namespace	new	not	not_eq	operator	or
or_eq	private	protected	public	register	reinterpret_cast
return	short	signed	sizeof	static	static_cast
struct	switch	template	this	throw	true
try	typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t	while
xor	xor_eq				

3. Literales constantes

   Se distingue entre números reales, números enteros y cadenas de caracteres.

```
<num_entero> ::= <digito> {<digito>}                        |
                 <digito_octal> {<digito_octal>} (u|U|l|L)|
                 <digito> {<digito_hex>} H
<num_real>   ::= <digito> {<digito>}.{<digito>} [<factor_esc>]
<factor_esc> ::= E (+|-|e) {<digito>}
```

Si el número lleva `u` como sufijo, es un número sin signo. Si lleva `L` como sufijo es de tamaño `long`. Todo número al que se le antepone el carácter 0 está expresado en base 8. Si se le antepone 0x, está expresado en base 16. Cuando se usa `E` en un literal constante real se dice que el número está expresado en punto flotante (notación científica).

Una cadena de caracteres (*string*) en C++ es una secuencia de caracteres encerrados entre comillas dobles (" "). Las constantes que representan un solo carácter se encierran entre comillas simples(' ').

4. Delimitadores

   Son símbolos (con o sin significado propio) que indican comienzo o fin de una entidad. Son elementos indivisibles: por ejemplo al escribir == se debe hacer sin ningún carácter intercalado.

```
{   }   [   ]   #   ##  (   )   <:  >:  <%  >%  %:%:
%:  ;   :   ?   ::  .   .*  +   -   *   /   %   ^
&   |   ~   !   =   <   >   +=  -=  *=  /=  %=  ^=
&=  |=  <<= >>= <<  >>  ==  !=  <=  >=  &&  ||  ++
--  ,   ->  ->* ...
```

5. Comentarios

   Un comentario es una secuencia de caracteres que es ignorada por el compilador. Se usan para documentar el programa, de manera que aunque no contribuyan a resolver el problema, sí ayudan a mejorar la comprensión del programa. Hay dos formas de incluir un comentario en C++. El texto que hay entre `//` y el fin de línea se considera un comentario. También es comentario el texto que se encuentra entre los símbolos `/*` y `*/`. Los comentarios de este estilo no se pueden anidar.

   Debe tenerse cuidado para no encerrar trozos de código útil dentro de un comentario, ya que un comentario puede extenderse a lo largo de múltiples líneas. Encerrar trozos de código como comentarios es útil cuando se está desarrollando un programa para comprobar si ciertas partes funcionan o no, pero es un error muy común encerrar por descuido otras zonas de código que son necesarias o bien olvidarse de sacar sentencias de dentro de un comentario.

6. Separadores

Son espacios en blanco, tabuladores, retornos de carro, fin de fichero y fin de línea. Habrá ocasiones en que se pongan para dar legibilidad al programa, y otras por necesidad. Los comentarios son considerados también separadores por el compilador.

2.2 Constantes, variables y tipos de datos

En un programa intervienen objetos sobre los que actúan las instrucciones que lo componen. Algunos de estos objetos tomarán valores a lo largo del programa. Dependiendo de si pueden cambiar de valor o no, podemos distinguir dos tipos de objetos:

- Constante: Objeto, referenciado mediante un identificador, que tomará un valor al principio (zona de declaraciones) y no se podrá modificar a lo largo del programa.
- Variable: Objeto, referenciado por un identificador, que puede tomar distintos valores a lo largo del programa.

Será misión del programador asignar el identificador que desee a cada constante y variable. El identificador o nombre de cada objeto sirve para identificar sin ningún tipo de ambigüedad a cada objeto, diferenciándolo de los demás objetos que intervienen en el programa.

En C++ hay que indicar el nombre de las constantes y variables que vamos a usar, para que el compilador pueda asociar internamente a dichos nombres las posiciones de memoria correspondientes. La declaración tiene que encontrarse en el código antes de cualquier instrucción que las use.

Al realizar las declaraciones de las variables, además de indicar su identificador o nombre, hay que indicar explícitamente el tipo de los datos que pueden contener.

Un tipo determina la clase de valores que puede asumir una variable, constante o expresión. Cada valor pertenece a un tipo. Sobre los tipos de datos hay que destacar:

- **Importancia de poseer tipo**. Cada variable, constante y expresión tienen asociado un único tipo. La información de tipo determina la representación de los valores, y la cantidad de espacio de memoria que debe serle asignado. Esa información también determina la forma en que deben ser interpretados los operadores aritméticos, y permite al compilador detectar errores semánticos en aquellos programas que contienen operaciones inapropiadas. El tipo de una variable no sólo determina el conjunto de valores que puede almacenar sino también el conjunto de operaciones permitidas sobre dicha variable. Esto último es el principio fundamental de la programación usando Tipos Abstractos de Datos (TADs).

- **Cardinalidad**. El número de valores que tiene un tipo de datos.

- **Operadores básicos**. A cada tipo se le asocia un conjunto de operadores básicos, destinados a ser de utilidad en el diseño de programas y que, además, tienen por lo menos una implementación razonablemente eficiente en un computador. Por supuesto, la selección de operadores básicos es, en cierta medida, arbitraria, y podría haberse aumentado o disminuido. El principio que suele seguirse es posibilitar al programador construir cualquier operación adicional en función de las básicas, y permitir hacerlo de una forma eficiente. Las operaciones más importantes y más generales definidas para los datos de cualquier tipo, son la asignación (representada por =) y la verificación de igualdad (==).

- **Compatibilidad de tipos**. Cada operador actúa sobre operandos de un mismo tipo, y da como resultado un valor siempre de un mismo tipo determinado. Como ya se ha dicho la información de tipo ayuda a los compiladores para detectar si se están realizando operaciones inapropiadas con tipos de datos distintos. Cuando el mismo símbolo es aplicado para varios tipos distintos (por ejemplo + para la suma de enteros y la de reales –se le llama operador sobrecargado–), dicho símbolo puede considerarse ambigüo, denotando varios operadores concretos diferentes.

En un lenguaje de programación existen normalmente una serie de tipos de datos predefinidos (tipos básicos) que el programador puede utilizar directamente para declarar variables. Por otro lado se pueden definir nuevos tipos propios para satisfacer las necesidades particulares del programador. Los principales tipos de datos predefinidos en C++ son:

- El tipo `int`. Números enteros.

- El tipo `bool`. Valores lógicos Verdadero y Falso.

- El tipo `char`. Caracteres del código ASCII.

- El tipo `float`. Números reales.

Todos ellos tienen dos propiedades en común: cada uno está formado por elementos indivisibles o atómicos que además están ordenados. Los tipos de datos con estas características se denominan tipos de datos escalares. Cuando decimos que un valor es atómico, nos referimos a que no contiene partes componentes a las que pueda acceder-se independientemente. Por ejemplo, un carácter es indivisible, mientras que la cadena

de caracteres "Hola"no. Cuando hablamos de que un conjunto de valores está ordenado, queremos decir que los operadores relacionales estándares (`<, >, ==, <=, >=, !=`) pueden aplicarse a aquéllos.

Todos estos tipos, excepto `float`, poseen una propiedad adicional. Cada valor tiene un *predecesor* y un *sucesor* únicos (excepto el primer y último valor, respectivamente). Los tipos de datos escalares que tienen esta propiedad se denominan tipos de datos ordinales. De los tipos de datos predefinidos sólo el tipo float no es ordinal, pues 0.0501 es el predecesor de 0.0502 con cuatro dígitos de precisión, pero 0.05019 es el predecesor de 0.05020 con un dígito más de precisión.

2.2.1 El tipo `int`

Los valores que pertenecen a este tipo son los números enteros. Los operadores aplicables a los enteros son:

- + suma
- - resta
- * producto
- / cociente de la division entera
- % resto de la división entera.

Cada computador restringirá el conjunto de valores de este tipo a un conjunto finito de enteros, comprendido en el intervalo -2^{N-1} ... $2^{N-1} - 1$, donde N es el número de bits que el computador usa para representar internamente un entero.

En nuestro caso, para el tipo `int`, N = 32, por lo que este rango es: - 2147483648 .. 2147483647. Se pueden definir variables para valores enteros con un rango menor indicando que son de tipo `short int`. En nuestro compilador, para los `short int` N = 16 y el rango es: -32768 .. 32767. Asimismo, `long int` está pensado para un rango mayor de valores, pero en el caso de nuestro compilador coincide con los `int`.

2.2.2 El tipo `unsigned int`

No es tipo independiente de `int`, sino una variación de aquél. Sus valores son los números enteros no negativos, esto es, los números naturales (incluido el 0). Los operadores aplicables son los mismos que para `int`. La ventaja de usar `unsigned int` en lugar de `int` es que hacemos explícito que una determinada variable sólo puede ser positiva.

Además de esto, si un computador usa N bits para almacenar un entero, el rango de los valores unsigned `int` será 0...2N - 1, con lo que podemos almacenar un número entero positivo más grande que con el tipo `int`. En nuestro caso los rangos para `unsigned int` y `short unsigned int` son, respectivamente: 0 .. 4294967295 y 0 .. 65535.

2.2.3 El tipo `float`

Los valores del tipo `float` son los números reales. Los operadores disponibles son los básicos anteriormente mencionados. La división se denota por /, siendo la división real, no la entera. Las constantes de tipo `float` se caracterizan por tener un punto decimal y, posiblemente, un factor de escala. Por ejemplo:

```
1.5    1.50    1.5E2    2.34E-2    0.0    0.    5.
```

Si se quieren representar números con mayor precisión, se puede usar el tipo `double` y `long double`.

En C++ el mayor real representable es 3.40282347e+38F (6 dígitos de precisión) y el menor número positivo distinto de 0 es 1.17549435e-38F.

2.2.4 El tipo `bool`

Un valor `bool` es uno de los dos valores de verdad lógicos denotados por los identificadores estándares del lenguaje: `true` y `false` (Verdadero y Falso). Las expresiones de tipo bool son muy importantes en programación. Estos valores no pueden leerse o escribirse directamente desde el teclado o al monitor, al contrario de lo que ocurre con los tipos numéricos.

Cuando se evalúa una expresión en C++, cualquier valor distinto de 0 es considerado `true` y 0 es considerado `false`.

2.2.5 El tipo `char`

Cada computador se comunica con su entorno mediante algún dispositivo de entrada y salida. Lee, escribe, o imprime elementos pertenecientes a un conjunto fijo de caracteres. Este conjunto constituye el rango de valores del tipo `char`. El conjunto fijo de caracteres más estandarizado es el denominado ASCII (*American Standard Code for Information Interchange*). Este conjunto está ordenado y cada carácter tiene una posición fija o número ordinal.

Las constantes de tipo char se denotan por el carácter encerrado entre comillas simples(' '). Un valor carácter puede ser asignado a una variable de tipo char, pero no puede

usarse en operaciones aritméticas. Sin embargo, las operaciones aritméticas pueden aplicarse a sus números obtenidos por la función de cambio de tipo `int(ch)`. De forma inversa el carácter con el número n asociado, se obtiene con la función estándar `char(n)`. Estas dos funciones se complementan se relacionan con las ecuaciones:

```
int(char(ch)) = ch    y    char(int(n)) = n
```

2.2.6 Chequeo de tipos

En C++ cada variable y constante tiene un tipo concreto, y cuando se efectúa una operación, sus operandos deben tener el tipo apropiado.

En principio, y hasta que no presentemos las reglas para el uso de tipos, supondremos que las operaciones sólo están definidas entre operandos del mismo tipo. Dos tipos son iguales si tienen el mismo nombre, y no si su composición es la misma. Esto resultará especialmente importante cuando se trabaje con tipos de datos estructurados.

Si el programador necesita usar expresiones en las que los operandos sean de distintos tipos, debe efectuar conversión de tipos explícita (casting). Para ello se especifica el tipo al que se convierte y, entre paréntesis, la expresión que se convierte.

```
int(Valor)    //Convierte el parametro Valor al tipo int.
float(Valor)  //Convierte el parametro Valor al tipo float.
etc...
```

2.3 Estructura de un programa en C++

Comenzaremos viendo la estructura global de un programa escrito en C++ para a continuación ir desglosando y analizando cada uno de sus componentes.

2.3.1 Esquema general de un programa

```
//lista de inclusiones
//declaraciones globales

int main(int argc, char *argv[])
{
    //declaraciones locales
    //sentencias de programa
}
```

2.3.2 Listas de inclusiones

Dentro de un programa podemos necesitar algunos procesos que por ser muy frecuentes están disponibles en el lenguaje y por tanto no nos debemos preocupar de programarlos. Para ello hay que indicar explícitamente que estamos incluyendo en nuestro programa los ficheros que contienen esos recursos. Una línea de inclusión consta de la expresión reservada `#include` y el nombre del fichero a incluir entre los caracteres `'<'` y `'>'`:

```
#include <cstdlib>
#include <iostream>
```

2.3.3 Espacios de nombres

Es posible que en varios ficheros diferentes se encuentre un recurso con el mismo nombre. Para evitar la confusión hay una herramienta, los *espacios de nombres*, que permite agrupar bajo un mismo contexto distintos recursos. Cuando se vaya a hacer uso de un recurso hay que especificar a qué espacio de nombre pertenece. Así podemos usar en un mismo programa dos recursos que, aunque tengan el mismo nombre, pertenezcan a espacios de nombres distintos. Existen tres formas de refenciar un recurso definido en un espacio de nombres:

- Nombrado explícito: Se escribe el identificador del espacio de nombre, seguido del los caracteres `::` y el identificador del recurso. Por ejemplo `std::cout`, nos permite usar el recurso `cout` definido en el espacio de nombres `std`. Más adelante en este mismo tema se explicará qué es `cout`.

- Importación explícita. Se declara en una línea de importación cuáles van a ser los recursos del espacio de nombres que se van a usar, mediante la sentencia `using`. Por ejemplo, si importamos `using std::cout;`, podemos usar en nuestro programa el recurso `cout` simplemente escribiendo su nombre, pero cualquier otro recurso del mismo espacio de nombres hay que referenciarlo junto con el identificador del espacio de nombres, por ejemplo `std::cin`.

- Importación general. Se importa con una sentencia `using` un espacio de nombres completo, por lo que se pueden usar todos sus recursos sin tener que hacer referencia al identificador del espacio de nombres. Por ejemplo si escribimos `using namespace std;`, basta con escribir `cout` y `cin` para acceder a esos recursos.

Nosotros no vamos a definir ningún espacio de nombres propio durante este curso y, de los que ya están definidos, sólo usaremos recursos pertenecientes al espacio de nombres

`std`, que es el que contiene todos los recursos *estandar* del lenguaje.

2.3.4 Declaraciones

En un programa escrito en C++ habrá que declarar todos los objetos que se empleen. Aunque C++ da una gran libertad sobre el lugar en que se pueden hacer nuevas declaraciones, nosotros nos vamos a restringir a hacerlo en ciertos lugares, bien antes del inicio de la función main para los objetos globales, bien antes de las instrucciones de una función en concreto para los objetos locales a una función.

2.3.5 Cuerpo del programa

Es la zona del programa formada por las sentencias a ejecutar, especificándose los pasos a seguir para la resolución del problema.

En C++ hay diversos tipos de sentencias:

Asignación	`switch`	`for`	`exit`
Llamada a función	`while`	`return`	`break`
`if`	`do-while`	`continue`	

Entre estas sentencias no hay ninguna dedicada a entrada/salida (E/S). Lo que ocurre es que la E/S en C++ se efectúa mediante llamadas a procedimientos que hay que importar mediante listas de inclusiones.

Los procedimientos que se importan se encuentran en bibliotecas. Aunque existe un conjunto estándar de bibliotecas, cada compilador de C++ puede añadir sus propias librerías predefinidas e informar de las características de los procedimientos que contiene. Éste es un aspecto a revisar siempre que, aún usando el mismo lenguaje, cambiemos de compilador.

```
<sec_sent>  ::=    <sentencia>; {<sentencia>}
<sentencia> ::=    <sent_asignacion>       |
                   <llamada_funcion>       |
                   <sent_if>               |
                   <sent_switch>           |
                   <sent_while>            |
                   <sent_do_while>         |
                   <sent_for>              |
                   exit                    |
                   break                   |
                   continue                |
```

```
                    return[<expresion>]     |
                    <sent_vacia>
```

2.3.6 Asignación

Es el operador que permite dar un valor a una variable.

```
<sent_asignacion> ::= <variable> = <expresion>
```

Ejemplo:

```
i = 10;
```

El operador de asignación es "=", y se lee *toma el valor*. La variable (izquierda) toma el valor de la expresión (derecha). Por defecto, las variables, después de haber sido declaradas, no tienen ningún valor (están indefinidas) y el programador debe encargarse de inicializarlas, es decir, asignarles un valor conocido, en el caso de que se vaya a utilizar su contenido.

En el lado izquierdo de la sentencia de asignación sólo puede haber una variable. Se evalúa la expresión de la derecha y el resultado se guarda en la variable.

Las variables mantienen el valor que se les asigna hasta que cambian mediante otra sentencia de asignación (u otro mecanismo), momento en que pierden el antiguo valor.

2.3.7 Expresiones

En los lenguajes de programación el concepto de expresión es similar al usado en matemáticas. Una expresión será un conjunto de operadores y operandos que tras su evaluación devolverá un único valor. Para saber el resultado de una expresión debemos conocer la precedencia de los operadores, es decir, el orden en el que se evalúan dentro de la misma. De mayor a menor precedencia, los operadores disponibles son:

```
!
*  /  %  &&
+  -  ||
== != <  >    <= >=
```

En caso de igualdad de precedencia se evalúan de izquierda a derecha. Estas reglas pueden alterarse mediante el uso de paréntesis.

Ejemplos:

```
A = 3 * B - 7  ↔  A = (3*B)-7
A = 3 - B - C  ↔  A = (3-B)-C
A = 3 - B * 7  ↔  A = 3-(B*7)
```

2.4 Entrada/salida en C++

Trataremos por el momento la entrada y salida de datos en forma de caracteres. En principio supondremos que nuestros programas de C++ son procesos (programas en ejecución) que toman información de un flujo de entrada y entregan información a un flujo de salida, utilizando para ello dispositivos externos capaces de manejar información en forma de caracteres organizados como una sucesión de líneas.

Se conocen como flujos de entrada/salida estándares aquellos que se realizan con la pantalla y con el teclado (E/S interactiva).

Otra alternativa de comunicación con un programa consiste en preparar un fichero de datos, por ejemplo en el disco, de donde el programa los pueda leer cuando los necesite o grabar cuando tenga resultados (E/S diferida).

En C++ las instrucciones para efectuar E/S se encuentran en módulos de librería, es decir, módulos preprogramados disponibles para el programador. A continuación se describirán algunas de las instrucciones de E/S interactiva contenidas en la librería `iostream`.

2.4.1 Salida

Para enviar datos a la salida por defecto (generalmente la pantalla del monitor del ordenador), vamos a usar el flujo de salida `cout`. Este flujo se encuentra en el fichero de biblioteca `iostream`. Aunque se pueden efectuar muchas operaciones diferentes sobre este flujo, por ahora sólo vamos a ver la operación `<<`. Esta operación manda un valor al flujo `cout`. Este valor puede ser de cualquier tipo. Ejemplo:

```
cout << "El valor de la variable cuyo nombre es i es ";
cout << i;
cout << endl;
```

El primer ejemplo escribe una cadena de caracteres, el segundo el contenido de la variable `i` y en el tercero se escribe el valor del carácter `endl`, lo que hace que se escriba una nueva línea. También se pueden concatenar los valores que se vayan a mandar a la salida. La siguiente instrucción equivale a las tres anteriores:

```
cout << "El valor de la variable cuyo nombre es i es " << i << endl;
```

2.4.2 Entrada

Este apartado se refiere a todas aquellas instrucciones que hacen posible introducir datos para que el programa los maneje. A continuación veremos instrucciones que también se encuentran en el fichero de biblioteca `iostream`. En este caso, el flujo de entrada en `cin` y la operación que vamos a ver de momento para la lectura de datos es `>>`. Como en el caso de la salida, el parámetro puede ser de cualquier tipo. Ejemplo:

```
int i, j;
char c;
cin >> i;
cin >> j;
cin >> c;
```

Finalmente vamos a señalar algunas notas importantes que hay que tener en cuenta al hacer entrada/salida interactiva:

1. Las entradas y salidas de nuestro programa tienen eco normalmente en el monitor y se mezclan. El programador debe prever este efecto para que el resultado sea legible.

2. El computador normalmente no se percata de lo que se teclea hasta pulsar la tecla `ENTER`, quien provoca además un salto de línea en la pantalla. Por tanto es posible una edición limitada de los valores que se están tecleando (borrarlos, cambiarlos, etc...) antes de pulsar `ENTER`.

3. Hay otras instrucciones que permiten hacer una salida con formato para que sea más visible, aunque por ahora no lo vamos a ver.

Ejercicios

1. Copiar (editar), compilar y ejecutar el siguiente programa que permite introducir una medida en centímetros y muestra la medida equivalente en pulgadas (una pulgada mide 2,54 centímetros).

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;

const float cent_por_pulgada = 2.54;

int main()
{
   float centimetros, pulgadas;

   cout << "Teclee una medida en centimetros\n";
   cin >> centimetros;
   pulgadas = centimetros / cent_por_pulgada; // Calculo de pulgadas
   cout << "Sus " << centimetros << " centimetros equivalen a " <<
           pulgadas << "pulgadas" << endl;

   return 0;
}
```

2. Escribir (editar), compilar y ejecutar el programa siguiente, que recibe el radio de un círculo y que muestra en pantalla el diámetro, la longitud de la circunferencia y el área de ese círculo.

```
#include <iostream>
#include <cstdlib>

using namespace std;

const float PI = 3.14;

int main() {
   float radio, diametro, longitud, area;
   cout << "Teclee la medida del radio de la circunferencia en cm\n";
   cin >> radio;
   // Calculos de magnitudes
   diametro = 2.0 * radio;
   longitud = PI * diametro;
```

```
   area = PI * radio * radio;
   // Salida de resultados
   cout << "DATOS DEL CIRCULO:" << endl;
   cout << "Radio introducido por el usuario: " << radio <<
           " cm" << endl;
   cout << "Diametro de la circunferencia: " << diametro <<
           " cm" << endl;
   cout << "Longitud de la circunferencia: " << longitud <<
           " cm" << endl;
   cout << "Area de la circunferencia: " << area << " cm2" << endl;

   return 0;
}
```

3. Al principio de un viaje en automóvil, el conductor se asegura de que el depósito de combustible esté lleno y anota la lectura del cuentakilómetros. Al terminar su viaje, anota la nueva lectura y la cantidad de combustible que se requirió para volver a llenar el depósito. Copiar el siguiente programa, que calcula el consumo de combustible (por cada 100 Km.), con una precisión de tres decimales, a partir de esta información.

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
   float kmInicial, kmFinal, litros, consumo;

   // Datos de entrada
   cout << "Teclee el kilometraje al principio del viaje" << endl;
   cin >> kmInicial;
   cout << "Teclee el kilometraje al final del viaje" << endl;
   cin >> kmFinal;
   cout << "Teclee los litros necesarios para llenar el deposito" << endl;
```

```
    cin >> litros;

    // Calculos de resultados
    consumo = 100.0 * litros / (kmFinal - kmInicial);
    // Salida de resultados
    cout << "El automovil consume " << consumo <<
            " litros por cada 100 km." << endl;

    return 0;
}
```

4. Escriba un programa que acepte un dato de tipo `int` de teclado y posteriormente lo escriba en pantalla. Ejecútelo introduciendo un número `int` válido, y posteriormente ejecútelo introduciendo por teclado un valor que no pertenezca al tipo `int`. Evalúe las diferencias entre ambas ejecuciones del mismo programa.

5. Escriba un programa que sólo declare variables de tipo `short int`. El programa deberá leer dos números desde el teclado, posteriormente los sumará, almacenando el resultado en una variable, y finalmente escribirá por pantalla el resultado de la suma. Ejecute dicho programa tomando como datos de entrada 1 y 90000. ¿Por qué no funciona?.

6. Escriba un programa que declare cuatro variables de tipo `int` y las escriba en pantalla sin haberles dado ningún valor. ¿Cuál es el resultado escrito? ¿Por qué?

7. Escribir un programa que reciba el precio neto de cierta mercancía comprada y la tasa de impuesto al valor añadido (IVA) como un porcentaje, y que muestre en pantalla el importe del IVA y el total a pagar en una forma correctamente anotada.

8. La distancia que recorre un objeto cuando se le deja caer desde cierta altura después de cierto tiempo se calcula mediante la siguiente fórmula de gravitación: $s = \frac{1}{2}gt^2$ con (g=9.81 m/s^2). Realizar un programa en C++ que pidiendo al usuario del programa el tiempo después de haberse soltado un objeto, calcúlense tanto la distancia recorrida como la velocidad media durante ese tiempo.

9. Escribir un programa que lea el lado de un cubo y calcule el área lateral y el volumen del mismo.

10. Escribir un programa para calcular el salario bruto semanal de un empleado, pidiendo por teclado la tarifa que se le paga por hora por su trabajo y el número de horas normales y extra trabajadas durante la semana. Las horas normales se pagan según la tarifa estándar y cualquier tiempo extra se paga a 1,5 veces la tarifa estándar por hora.

11. Escriba un programa que intercambie el valor de dos variables enteras. Posteriormente amplíelo para que intercambie el valor de tres variables. Si inicialmente los valores eran $\{x == x_0; y == y_0; z == z_0\}$, al final del programa los valores deben ser $\{x == y_0; y == z_0; z == x_0\}$. Opcional: Haga una nueva versión el ejercicio sin usar variables intermedias.

12. Escriba un programa que transforme un número dentro de un intervalo $[x_0, y_0]$ al punto equivalente dentro de otro intervalo $[x_1, y_1]$.

13. Escriba un programa en el que se asigna cada una de las siguientes fórmulas matemáticas a una variable real.

$$3x$$
$$5x + 6y$$
$$\frac{x+7}{y}$$
$$\frac{x+y}{z*3}$$

14. Escriba un programa que lea una palabra de cuatro letras por teclado, y posteriormente escriba dicha palabra de manera que cada letra se encuentre codificada sustituyéndola por aquel carácter que le sigue en la tabla de código ASCII.

15. Escriba un programa que lea una palabra de cuatro letras y a continuación la escriba en mayúsculas. Hágalo primero usando la función `toupper` y después haga una nueva versión sin usar la función `toupper`. La función `toupper` se encuentra en el fichero de biblioteca `cctype`.