Capítulo 1: Fundamentos de C++

Desarrollo y Propiedades de C++

Características

- **C++**
  - **C**
    - Universal
    - Eficiente
    - Cercano a la máquina
    - Portable
  - **OOP**
    - Abstracción de datos
    - Ocultamiento de datos
    - Herencia
    - Polimorfismo
  - **Extensiones**
    - Manejo de excepciones
    - Plantillas / *Templates*

Características de C++

C++ no es un lenguaje puramente orientado a objetos, sino un híbrido que contiene la funcionalidad del lenguaje de programación C. Esto significa que tiene todas las características que están disponibles en C:

- Programas modulares de uso universal
- Programación eficiente, cercana a la máquina
- Programas portables para varias plataformas.

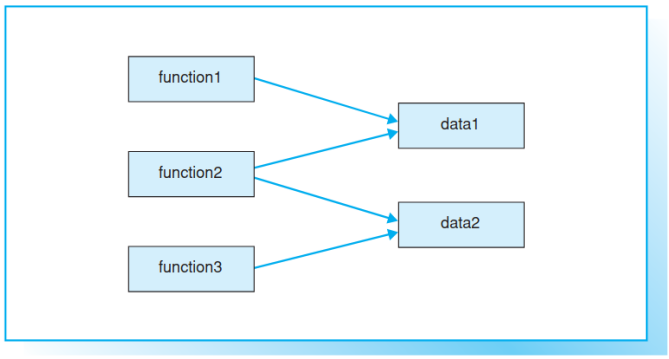
Las grandes cantidades de código fuente C existente también se pueden utilizar en programas C++. C++ admite los conceptos de programación orientada a objetos (o OOP para abreviar), que son:

- *Abstracción de datos*, es decir, la creación de clases para describir objetos
- *Encapsulación de datos* para acceso controlado a datos de objetos
- *Herencia* mediante la creación de clases derivadas (incluidas múltiples clases derivadas)
- *Polimorfismo* (del griego multiforme), es decir, la implementación de instrucciones que pueden tener efectos variables durante la ejecución del programa.

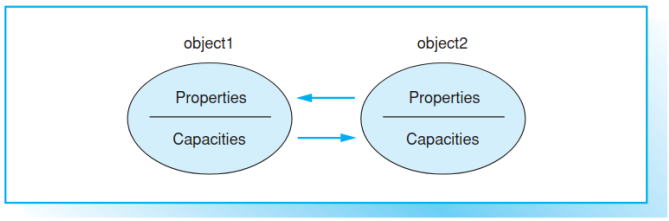
Se agregaron varios elementos del lenguaje C++, como referencias, plantillas y manejo de excepciones. Si bien estos elementos del lenguaje no son estrictamente funciones de programación orientada a objetos, son importantes para la implementación eficiente de programas.

Programación Orientada a Objetos

Concepto tradicional



Concepto Orientado a Objetos



Programación Procedimental Tradicional

En la programación procedimental tradicional, los datos y las funciones (subrutinas, procedimientos) se mantienen separados de los datos que procesan. Esto tiene un efecto significativo en la forma en que un programa maneja los datos:

- El programador debe asegurarse de que los datos se inicialicen con valores adecuados antes de su uso y de que los datos adecuados se pasen a una función cuando se la llama
- Si se cambia la representación de los datos, por ejemplo, si se extiende un registro, también se deben modificar las funciones correspondientes.

Ambos puntos pueden conducir a errores y ninguno de ellos admite requisitos de mantenimiento de programa bajos.

Objetos

La programación orientada a objetos desplaza el foco de atención a los *objetos*, es decir, a los aspectos en los que se centra el problema. Un programa diseñado para mantener cuentas bancarias trabajaría con datos como saldos, límites de crédito, transferencias, cálculos de intereses, etc. Un objeto que represente una cuenta en un programa tendrá propiedades y capacidades que son importantes para la gestión de cuentas. Los objetos OOP combinan datos (propiedades) y funciones (capacidades). Una clase define un cierto tipo de objeto definiendo tanto las propiedades como las capacidades de los objetos de ese tipo. Los objetos se comunican enviándose "mensajes" entre sí, que a su vez activan las capacidades de otro objeto.

Ventajas de la programación orientada a objetos

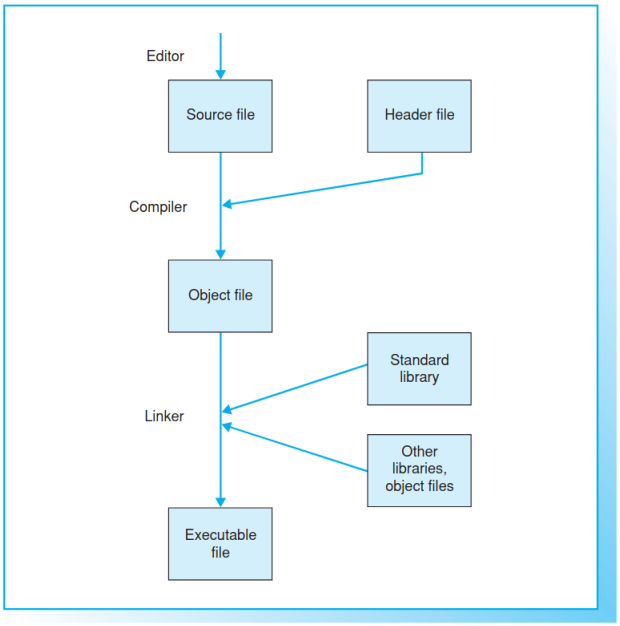
La programación orientada a objetos ofrece varias ventajas importantes para el desarrollo de software:

- **Menor susceptibilidad a errores**: un objeto controla el acceso a sus propios datos. Más específicamente, un objeto puede rechazar intentos de acceso erróneos

- **Fácil reutilización**: los objetos se mantienen a sí mismos y, por lo tanto, pueden usarse como bloques de construcción para otros programas
- **Bajo requerimiento de mantenimiento**: un tipo de objeto puede modificar su propia representación de datos internos sin requerir cambios en la aplicación.

Desarrollando un programa C++

Traducción de un programa C++



Para crear y traducir un programa C++ se requieren los siguientes tres pasos:

1. Primero, se utiliza un editor de texto para guardar el programa C++ en un archivo de texto. En otras palabras, el *código fuente/source code* se guarda en un *archivo fuente/source file*. En proyectos más grandes, el programador normalmente utilizará la *programación modular*. Esto significa que el código fuente se almacenará en varios archivos fuente que se editan y traducen por separado.
2. El archivo fuente se pasa por un compilador para su traducción. Si todo funciona como se ha planeado, se crea un archivo objeto compuesto por código de máquina. El archivo objeto también se denomina *módulo*.
3. Finalmente, el enlazador combina el archivo objeto con otros módulos para formar un *archivo ejecutable/executable file*. Estos módulos adicionales contienen funciones de bibliotecas estándar o partes del programa que se han compilado previamente.

Es importante utilizar la extensión de archivo correcta para el nombre del archivo fuente. Aunque la extensión del archivo depende del compilador que utilice, las extensiones de archivo más comunes son `.cpp` y `.cc`.

Antes de la compilación, los *archivos de encabezado/header files*, también conocidos como *archivos de inclusión/include files*, se pueden copiar al archivo fuente. Los archivos de encabezado son archivos de texto que contienen información necesaria para varios archivos fuente, por ejemplo, definiciones de tipos o declaraciones de variables y funciones. Los archivos de encabezado pueden tener la extensión de archivo `.h`, pero es posible que no tengan ninguna extensión de archivo.

La *biblioteca estándar/standard library* de C++ contiene funciones predefinidas y estandarizadas que están disponibles para cualquier compilador.

Los compiladores modernos normalmente ofrecen un *entorno de desarrollo de software integrado*, que combina los pasos mencionados anteriormente en una sola tarea. Hay disponible una interfaz gráfica de

usuario para editar, compilar, vincular y ejecutar la aplicación. Además, se pueden iniciar herramientas adicionales, como un depurador.

Nota

Si el archivo fuente contiene solo un *error de sintaxis*, el compilador informará un error. Es posible que se muestren mensajes de error adicionales si el compilador intenta continuar a pesar de haber encontrado un error. Por lo tanto, cuando esté solucionando problemas con un programa, asegúrese de comenzar con el primer error que se muestre.

Además de los mensajes de error, el compilador también emitirá *advertencias/warnings*. Una advertencia no indica un error de sintaxis, sino que simplemente llama su atención sobre un posible error en la lógica del programa, como el uso de una variable no inicializada.

Programa en C++ para principiantes

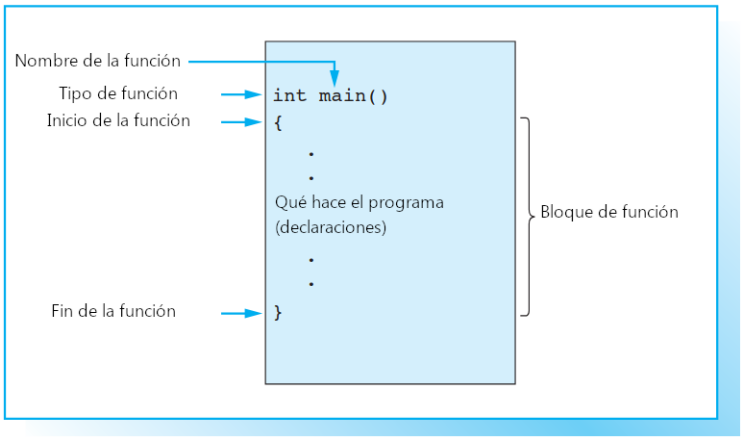
Programa de ejemplo

```
#include <iostream>
using namespace std;

int main()
{
    cout << "¡Diviértete con C++!" << endl;
    return 0;
}
```

Salida de pantalla

```
¡Diviértete con C++!
```

Estructura de la función `main()`



Un programa en C++ está formado por objetos con sus *funciones miembro* y *funciones globales* que los acompañan, que no pertenecen a ninguna clase en particular. Cada función cumple su propia tarea particular y también puede llamar a otras funciones. Puede crear funciones usted mismo o utilizar funciones ya preparadas de la biblioteca estándar. Siempre necesitará escribir usted mismo la función global `main()`, ya que tiene un papel especial que desempeñar; de hecho, es el programa principal.

El breve ejemplo de programación de la página opuesta demuestra dos de los elementos más importantes de un programa en C++. El programa contiene solo la función `main()` y muestra un mensaje.

La primera línea comienza con el símbolo de número, `#`, que indica que la línea está destinada al *preprocesador*. El preprocesador es solo un paso en la primera fase de traducción y no se crea ningún código objeto en este momento. Puede escribir

```
#include <filename>
```

para que el preprocesador copie el archivo citado en esta posición en el código fuente. Esto permite al programa acceder a toda la información contenida en el archivo de encabezado. El archivo de encabezado `iostream` comprende convenciones para flujos de entrada y salida. La palabra *stream* indica que la información involucrada será tratada como un flujo de datos.

Los nombres predefinidos en C++ se encuentran en el espacio de nombres `std` (estándar). La directiva using permite el acceso directo a los nombres del espacio de nombres `std`.

La ejecución del programa comienza con la primera instrucción en la función `main()`, y es por eso que cada programa C++ debe tener una función principal. La estructura de la función se muestra en la página opuesta. Aparte del hecho de que el nombre no se puede cambiar, la estructura de esta función no es diferente de la de cualquier otra función C++.

En nuestro ejemplo, la función `main()` contiene dos *declaraciones/statements*. La primera declaración

```
cout << "¡Diviértete con C++!" << endl;
```

muestra la cadena de texto `¡Diviértete con C++!` en la pantalla. El nombre `cout` (salida de consola) designa un objeto responsable de la salida.

Los dos símbolos menor que, <<, indican que se están "enviando" caracteres al flujo de salida. Finalmente, `endl` (fin de línea) provoca un salto de línea. La declaración

```
return 0;
```

Termina la función `main()` y también el programa, devolviendo un valor de 0 como código de salida al programa que lo llamó. Es una práctica estándar usar el código de salida 0 para indicar que un programa ha terminado correctamente.

Tenga en cuenta que las instrucciones van seguidas de un punto y coma. Por cierto, la instrucción más corta solo incluye un punto y coma y no hace nada.

Estructura de programas simples en C++

Un programa en C++ con varias funciones

```
/*****************************************************
A program with some functions and comments
*****************************************************/

#include <iostream>
using namespace std;

void line(), message();   // Prototypes

int main()
{
    cout << "Hola! El programa comienza en main()."
         << endl;
    line();
    message();
    line();
    cout << "Al final de main()." << endl;

    return 0;
}

void line()               // Trazar una línea.
{
    cout << "---------------------------------" << endl;
}

void message()            // Para mostrar un mensaje.
{
    cout << "En función message()." << endl;
}
```

Salida de pantalla

```
¡Hola! El programa comienza en main().
-----------------------------------
En función message().
-----------------------------------
Al final de main().
```

El ejemplo de la página opuesta muestra la estructura de un programa C++ que contiene múltiples funciones. En C++, no es necesario definir las funciones en un orden fijo. Por ejemplo, podría definir primero la función `message()`, seguida de la función `line()` y, por último, la función `main()`.

Sin embargo, es más común comenzar con la función `main()`, ya que esta función controla el flujo del programa. En otras palabras, `main()` invoca funciones que aún no se han definido. Esto es posible si se proporciona al compilador un prototipo de función que incluye toda la información que necesita el compilador.

Este ejemplo también presenta comentarios. Las cadenas encerradas en `/* . . . */` o que comienzan con `//` se interpretan como comentarios.

Ejemplos:

```
/* Puedo cubrir
varias lineas */

// Puedo cubrir sólo una línea
```

En los comentarios de una sola línea, el compilador ignora los caracteres que siguen a los signos `//` hasta el final de la línea. Los comentarios que cubren varias líneas son útiles para solucionar problemas, ya que puede utilizarlos para enmascarar secciones completas de su programa. Ambos tipos de comentarios se pueden utilizar para comentar el otro tipo.

En cuanto al *diseño/layout* de los archivos fuente, el compilador analiza cada archivo fuente secuencialmente, descomponiendo el contenido en tokens, como nombres de funciones y operadores. Los tokens se pueden separar con cualquier número de caracteres de espacio en blanco, es decir, con espacios, tabulaciones o caracteres de nueva línea. El orden del código fuente es importante, pero no es importante adherirse a un diseño específico, como organizar el código en filas y columnas. Por ejemplo

```
void message
    (  ){ cout <<
      "En función message()." <<
endl;}
```

Puede resultar difícil de leer, pero es una definición correcta de la función `message()`.

Las directivas del preprocesador son una excepción a la regla de diseño, ya que siempre ocupan una sola línea. El signo de número, `#`, al principio de una línea solo puede ir precedido de un espacio o un carácter de tabulación.

Para mejorar la legibilidad de sus programas en C++, debe adoptar un estilo coherente, utilizando sangrías y líneas en blanco para reflejar la estructura de su programa. Además, haga un uso generoso de los comentarios.

Ejercicios

1. Escriba un programa en C++ que muestre el siguiente texto en la pantalla:

   ```
   ¡Oh, qué
   día tan feliz!
   ¡Oh, sí,
   qué día tan feliz!
   ```

   Utilice el manipulador `endl` cuando sea apropiado.

2. El siguiente programa contiene varios errores:

   ```
   */ Ahora no deberías olvidarte de tus gafas //
   #include <stream>
   int main
   {
       cout << "Si este texto",
       cout >> " aparece en tu pantalla, ";
       cout << " endl;"
       cout << 'Puedes darte una palmadita en la espalda '
            << "¡la parte de atrás! " << endl.
       return 0;
   )
   ```

Resuelva los errores y ejecute el programa para probar los cambios.

3. ¿Qué muestra en pantalla el siguiente programa C++?

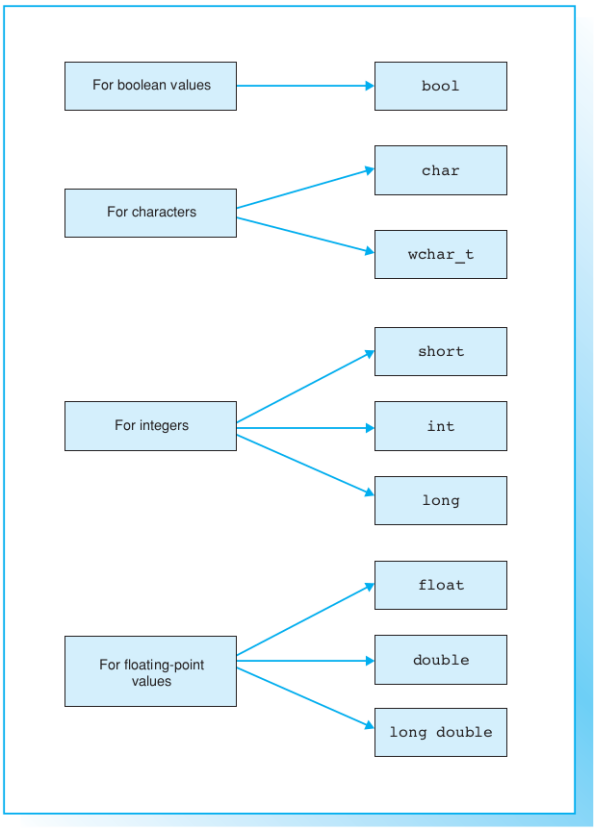
```
#include <iostream>
using namespace std;

void pause();      // Prototype

int main()
{
    cout << endl << "Estimado lector, "
         << endl << "tener un ";
    pause();
    cout << "!" << endl;

    return 0;
}

void pause()
{
    cout << "BREAK";
}
```

Capítulo 2: Tipos fundamentales, Constantes y Variables

En este capítulo se presentan los tipos y objetos básicos que utilizan los programas de C++.

Tipos fundamentales

Descripción general[1]



Un programa puede utilizar varios datos para resolver un problema dado, por ejemplo, caracteres, números enteros o números de punto flotante. Dado que una computadora utiliza diferentes métodos para procesar y guardar datos, se debe conocer el tipo de datos. El tipo define

1. La representación interna de los datos y
2. La cantidad de memoria que se debe asignar.

Un número como `-1000` se puede almacenar en 2 o 4 bytes. Al acceder a la parte de la memoria en la que se almacena el número, es importante leer el número correcto de bytes. Además, el contenido de la memoria, es decir, la secuencia de bits que se lee, se debe interpretar correctamente como un entero con signo.

El compilador de C++ reconoce los *tipos fundamentales*, también conocidos como tipos integrados, que se muestran en la página opuesta, en los que se basan todos los demás tipos (vectores, punteros, clases, ...).

El tipo `bool`

El resultado de una comparación o una asociación lógica utilizando `AND` u `OR` es un valor *booleano*, que puede ser verdadero o falso. C++ utiliza el tipo `bool` para representar valores booleanos. Una expresión del tipo `bool` puede ser verdadera o falsa, donde el valor interno de *verdadero* se representará como el valor numérico **1** y *falso* como un **cero**.

[1] Sin el tipo `void`, que se presentará más adelante.

Los tipos `char` y `wchar_t`

Estos tipos se utilizan para guardar códigos de caracteres. Un *código de carácter* es un entero asociado a cada carácter. La letra A se representa con el código 65, por ejemplo. El *conjunto de caracteres/character set* define qué código representa un determinado carácter. Al mostrar caracteres en pantalla, se transmiten los códigos de caracteres aplicables y el "receptor", es decir, la pantalla, es responsable de interpretar correctamente los códigos.

El lenguaje C++ no estipula ningún conjunto de caracteres en particular, aunque en general se utiliza un conjunto de caracteres que contiene el *código ASCII* (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Este código de 7 bits contiene definiciones para 32 caracteres de control (códigos 0 - 31) y 96 caracteres imprimibles (códigos 32 - 127).

El tipo *char* (carácter) se utiliza para almacenar códigos de caracteres en un byte (8 bits). Esta cantidad de almacenamiento es suficiente para conjuntos de caracteres extendidos, por ejemplo, el conjunto de caracteres ANSI que contiene los códigos ASCII y caracteres adicionales como las diéresis alemanas.

El tipo *wchar_t* (tipo de carácter ancho) comprende al menos 2 bytes (16 bits) y, por lo tanto, es capaz de almacenar caracteres Unicode modernos. *Unicode* es un código de 16 bits que también se utiliza en Windows NT y que contiene códigos para aproximadamente 35.000 caracteres en 24 idiomas.

Tipos integrales

Tipo	Tamaño	Rango de valores (decimal)
`char`	1 byte	-128 to +127 or 0 to 255
`unsigned char`	1 byte	0 to 255
`signed char`	1 byte	-128 to +127
`int`	2 byte resp.	-32768 to +32767 resp.
	4 byte	-2147483648 to +2147483647
`unsigned int`	2 byte resp.	0 to 65535 resp.
	4 byte	0 to 4294967295
`short`	2 byte	-32768 to +32767
`unsigned short`	2 byte	0 to 65535
`long`	4 byte	-2147483648 to +2147483647
`unsigned long`	4 byte	0 to 4294967295

Programa ejemplo

```
#include <iostream>
#include <climits>          // Definition of INT_MIN, ...
using namespace std;

int main()
{
    cout << "Range of types int and unsigned int"
         << endl << endl;
    cout << "Type          Minimum          Maximum"
         << endl
         << "-----------------------------------------------"
         << endl;

    cout << "int         " << INT_MIN << "          "
                           INT_MAX << endl;"

    cout << "unsigned int" << "          0          "
                           << UINT_MAX << endl;"

    return 0;
}
```

Tipos integrales

Los tipos `short`, `int` y `long` están disponibles para operaciones con números enteros. Estos tipos se distinguen por sus rangos de valores. La tabla muestra los tipos enteros, que también se denominan *tipos integrales*, con sus requisitos de almacenamiento típicos y rangos de valores.

El tipo `int` (entero) está hecho a medida para computadoras y se adapta a la longitud de un registro en la computadora. Para computadoras de 16 bits, int es equivalente a `short`, mientras que para

computadoras de 32 bits int será equivalente a `long`.

C++ trata los códigos de caracteres como números enteros normales. Esto significa que puede realizar cálculos con variables que pertenecen a los tipos `char` o `wchar_t` exactamente de la misma manera que con las variables de tipo `int`. El `char` es un tipo integral con un tamaño de un byte. El rango de valores es, por lo tanto, de -128 a +127 o de 0 a 255, dependiendo de si el compilador interpreta el tipo `char` como con signo o sin signo. Esto puede variar en C++.

El tipo `wchar_t` es otro tipo integral y normalmente se define como `unsigned short`.

Los modificadores `signed` y `unsigned`

Los tipos `short`, `int` y `long` normalmente se interpretan como con signo, y el bit más alto representa el signo. Sin embargo, los tipos integrales pueden ir precedidos de la palabra clave *unsigned*. La cantidad de memoria necesaria permanece inalterada, pero el rango de valores cambia debido a que el bit más alto ya no es necesario como signo. La palabra clave `unsigned` se puede utilizar como abreviatura de `unsigned int`.

El tipo `char` también se interpreta normalmente como con signo. Dado que se trata de una mera convención y no es obligatorio, la palabra clave `signed` está disponible. Por lo tanto, hay tres tipos disponibles: `char`, `signed char` y `unsigned char`.

Nota

En ANSI C++, el tamaño de los tipos enteros no está preestablecido. Sin embargo, se aplica el siguiente orden:

```
char <= short <= int <= long
```

Además, el tipo `short` comprende al menos 2 bytes y el tipo `long` al menos 4 bytes.

Los rangos de valores actuales están disponibles en el archivo de encabezado `climits`. Este archivo define constantes como `CHAR_MIN`, `CHAR_MAX`, `INT_MIN` e `INT_MAX`, que representan los valores más pequeños y más grandes posibles. El programa de la página opuesta genera el valor de estas constantes para los tipos `int` y `unsigned int`.

Tipos de punto flotante

Tipo	Tamaño	Rango de valores	Valor positivo más bajo	Precisión (decimal)
`float`	4 bytes	-3.4E+38	1.2E-38	6 dígitos
`double`	8 bytes	-1.7E+308	2.3E-308	15 dígitos
`long double`	10 bytes	-1.1E+4932	3.4E-4932	19 dígitos

Nota

El formato IEEE (IEEE = **I**nstitute of **E**lectrical and **E**lectronic **E**ngineers) se utiliza normalmente para representar tipos de punto flotante. La tabla anterior utiliza esta representación.

Tipos aritméticos

Tipos integrales

```
bool
char, signed char, unsigned char, wchar_t
short, unsigned short
int, unsigned int
long, unsigned long
```

Tipos de punto flotante

```
float
double
long double
```

Nota

Los operadores aritméticos se definen para tipos aritméticos, es decir, se pueden realizar cálculos con variables de este tipo.

Tipos de coma flotante

Los números con una parte fraccionaria se indican mediante un punto decimal en C++ y se denominan números de coma flotante. A diferencia de los números enteros, los números de coma flotante se deben almacenar con una precisión preestablecida. Los siguientes tres tipos están disponibles para los cálculos que involucran números de coma flotante:

`float` Para una precisión sencilla

`double` Para doble precisión

`long double` Para alta precisión

El rango de valores y la precisión de un tipo se derivan de la cantidad de memoria asignada y la representación interna del tipo.

La *precisión/accuracy* se expresa en decimales. Esto significa que "seis decimales" permiten a un programador almacenar dos números de punto flotante que difieren dentro de los primeros seis decimales como números separados. A la inversa, no hay garantía de que las cifras 12.3456 y 12.34561 se distingan cuando se trabaja con una precisión de seis decimales. Y recuerde, no es una cuestión de la posición del punto decimal, sino simplemente de la secuencia numérica.

Si es importante para su programa mostrar números de punto flotante con una precisión compatible con una máquina en particular, debe consultar los valores definidos en el archivo de encabezado `cfloat`.

Los lectores interesados en material adicional sobre este tema deben consultar el Apéndice, que contiene una sección sobre la representación de números binarios en computadoras tanto para números enteros como de punto flotante.

El operador `sizeof`

La cantidad de memoria necesaria para almacenar un objeto de un tipo determinado se puede determinar utilizando el operador `sizeof`:

```
sizeof(name)
```

da como resultado el tamaño de un objeto en bytes y el nombre del parámetro indica el tipo de objeto o el objeto en sí. Por ejemplo, `sizeof(int)` representa un valor de 2 o 4 según la máquina. Por el contrario, `sizeof(float)` siempre será igual a 4.

Clasificación

Los tipos fundamentales en C++ son los *tipos enteros*, los tipos de *punto flotante* y el tipo *void*. Los tipos utilizados para números enteros y de punto flotante se denominan colectivamente *tipos aritméticos*, ya que se definen operadores aritméticos para ellos.

El tipo *void* se utiliza para expresiones que no representan un valor. Por lo tanto, una llamada de función puede adoptar un tipo `void`.

Constantes

Ejemplos de constantes integrales

Decimal	Octal	Hexadecimal	Tipo
16	020	0x10	`int`
255	0377	Oxff	`int`
32767	077777	0x7FFF	`int`
32768U	0100000U	0x8000U	`unsigned int`
100000	0303240	0x186A0	`int (32 bit-)` `long (16 bit-CPU)`
10L	012L	0xAL	`long`
27UL	033UL	0x1bUL	`unsigned long`
2147483648	020000000000	0x80000000	`unsigned long`

Nota

En cada línea de la tabla anterior, el mismo valor se presenta de una manera diferente.

Programa de ejemplo

```
// Para mostrar literales enteros hexadecimales y
// literales enteros decimales.
//

#include <iostream>
using namespace std;

int main()
{
    // cout outputs integers as decimal integers:
    cout << "Value of 0xFF = " << 0xFF << " decimal"
         << endl;                    // Output: 255 decimal
    // El manipulador hexadecimal cambia la salida a hexadecimal
    // format (dec changes to decimal format):
    cout << "Value of 27 = " << hex << 27 <<" hexadecimal"
         << endl;                    // Output: 1b hexadecimal

    return 0;
}
```

Las palabras clave booleanas `true` y `false`, un número, un carácter o una secuencia de caracteres (cadena) son todas constantes, que también se denominan literales. Por lo tanto, las constantes se pueden subdividir en

- *Constantes booleanas*
- *Constantes numéricas*
- *Constantes de caracteres*
- *Constantes de cadena*

Cada constante representa un valor y, por lo tanto, un tipo, al igual que cada expresión en C++. El tipo se define por la forma en que se escribe la constante.

Constantes booleanas

Una expresión booleana puede tener dos valores que se identifican con las palabras clave `true` y `false`. Ambas constantes son del tipo `bool`. Se pueden utilizar, por ejemplo, para establecer indicadores que representan solo dos estados.

Constantes integrales

Las *constantes numéricas integrales* se pueden representar como números decimales simples, octales o hexadecimales:

- Una *constante decimal* (base 10) comienza con un número decimal distinto de cero, como `109` o `987650`
- Una *constante octal* (base 8) comienza con un 0 inicial, por ejemplo `077` o `01234567`
- Una *constante hexadecimal* (base 16) comienza con el par de caracteres `0x` o `0X`, por ejemplo `0x2A0` o `0X4b1C`. Los números hexadecimales pueden estar en mayúsculas o no.

Las constantes integrales normalmente son de tipo `int`. Si el valor de la constante es demasiado grande para el tipo `int`, se aplicará un tipo capaz de representar valores mayores. La clasificación de las constantes decimales es la siguiente:

`int`, `long`, `unsigned long`

Puede designar el tipo de una constante adicionando la letra `L` o `l` (para `long`), o `U` o `u` (para `unsigned`). Por ejemplo,

`12L`	and	`12l`	corresponde al tipo `long`
`12U`	and	`12u`	corresponde al tipo `unsigned int`
`12UL`	and	`12ul`	corresponden al tipo `unsigned long`

Ejemplos de constantes de punto flotante

```
5.19            12.             0.75            0.00004
0.519E1         12.0            .75             0.4e-4
0.0519e2        .12E+2          7.5e-1          .4E-4
519.OE-2        12e0            75E-2           4E-5
```

Ejemplos de constantes de caracteres

```
Constante          Carácter                  Valor constante
                                             (código ASCII decimal)
-------------------------------------------------------------------
   'A'             Capital A                      65
   'a'             Lowercase a                    97
   ' '             Blank                          32
   '.'             Dot                            46
   '0'             Digit 0                        48
  '\0'             Carácter nulo de terminación   0
```

Representación interna de un literal de cadena

Literal de cadena: `"Hello!"`

Stored byte sequence:

'H'	'e'	'l'	'l'	'o'	'!'	'\0'

Constantes de punto flotante

Los números de punto flotante siempre se representan como decimales, utilizándose un punto decimal para distinguir la parte fraccionaria de la parte entera. Sin embargo, también se permite la notación exponencial.

Ejemplos:

```
27.1 1.8E-2 // Tipo: double
```

Aquí, `1.8E-2` representa un valor de $1{,}8 * 10^{-2}$. `E` también se puede escribir con una letra e minúscula. Siempre se debe utilizar un punto decimal o `E (e)` para distinguir las constantes de punto flotante de las constantes enteras.

Las constantes de punto flotante son de tipo `double` por defecto. Sin embargo, puede añadir `F` o `f` para designar el tipo `float`, o añadir `L` o `l` para el tipo `long double`.

Constantes de carácter

Una constante de carácter es un carácter encerrado entre *comillas simples*. Las constantes de carácter toman el tipo `char`.

Ejemplo:

```
'A' // Tipo: char
```

El valor numérico es el código de carácter que representa el carácter. La constante `'A'` tiene un valor de 65 en código ASCII.

Constantes de cadena

Ya conoce las constantes de cadena, que se introdujeron para la salida de texto mediante el flujo cout. Una constante de cadena consiste en una secuencia de caracteres encerrados entre *comillas dobles*.

Ejemplo:

```
"¡Hoy es un día hermoso!"
```

Una constante de cadena se almacena internamente sin las comillas pero termina con un *carácter nulo*, `\0`, representado por un byte con un valor numérico de 0, es decir, todos los bits de este byte se establecen en 0. Por lo tanto, una cadena ocupa un byte más en la memoria que el número de caracteres que contiene. Una *cadena vacía*, "", ocupa un solo byte.

El carácter nulo de terminación `\0` no es el mismo que el número cero y tiene un código de carácter diferente al cero. Por lo tanto, la cadena

Ejemplo:
"0"

comprende **dos** bytes, el primer byte contiene el código para el carácter cero 0 (código ASCII 48) y el segundo byte el valor 0.

El carácter nulo de terminación `\0` es un ejemplo de una secuencia de escape. Las secuencias de escape se describen en la siguiente sección.

Secuencias de escape

Descripción general

Carácter único	Significado	Código ASCII (decimal)
`\a`	alert (BEL)	7
`\b`	backspace (BS)	8
`\t`	horizontal tab (HT)	9
`\n`	line feed (LF)	10
`\v`	vertical tab (VT)	11
`\f`	form feed (FF)	12
`\r`	carriage return (CR)	13
`\"`	"(double quote)	34
`\'`	' (single quote)	39
`\?`	? (question mark)	63
`\\`	\(backslash)	92
`\0`	carácter de terminación de cadena	0
`\ooo` (up to 3 octal digits)	valor numérico de un carácter	ooo (octal!)
`\xhh` (hexadecimal digits)	valor numérico de un carácter	hh (hexadecimal!)

Programa de muestra

```
#include <iostream>
using namespace std;

int main()
{
cout << "\nThis is\t a string\n\t\t"
        " with \"many\" escape sequences!\n";

return 0;
}
```

Salida del programa:

```
This is    a string
           with "many" escape sequences!
```

Uso de caracteres especiales y de control

Los caracteres no gráficos se pueden expresar mediante *secuencias de escape*, por ejemplo, `\t`, que representa una tabulación.

El efecto de una secuencia de escape dependerá del dispositivo en cuestión. La secuencia `\t`, por ejemplo, depende de la configuración del ancho de la tabulación, que por defecto es de ocho espacios en blanco, pero puede tener cualquier valor.

Una secuencia de escape siempre comienza con una \(barra invertida) y representa un solo carácter. La tabla de la página opuesta muestra las secuencias de escape estándar, sus valores decimales y sus efectos.

Puede utilizar secuencias de escape octales y hexadecimales para crear cualquier código de carácter. Por tanto, la letra `A` (decimal 65) en el código ASCII también se puede expresar como `\101` (tres octales) o `\x41` (dos hexadecimales). Tradicionalmente, las secuencias de escape se utilizan únicamente para representar caracteres no imprimibles y caracteres especiales. Las secuencias de control para los controladores de pantalla e impresora se inician, por ejemplo, con el carácter ESC (decimal 27), que se puede representar como `\33` o `\x1b`.

Las secuencias de escape se utilizan en constantes de caracteres y cadenas.

Ejemplos:

```
'\t' "\t¡Hola\n\tMike!"
```

Los caracteres ', ", y \no tienen un significado especial cuando van precedidos de una barra invertida, es decir, se pueden representar como `\'`, `\"`, y `\\` respectivamente.

Cuando utilice números octales para secuencias de escape en cadenas, asegúrese de utilizar tres dígitos, por ejemplo, `\033` y no \33. Esto ayuda a evitar que los números posteriores se evalúen como parte de la secuencia de escape. No existe un número máximo de dígitos en una secuencia de escape hexadecimal. La secuencia de números hexadecimales termina automáticamente con el primer carácter que no sea un número hexadecimal válido.

El programa de ejemplo de la página opuesta demuestra el uso de secuencias de escape en cadenas. El hecho de que una cadena pueda ocupar dos líneas es otra característica nueva. Las constantes de cadena separadas únicamente por espacios en blanco se concatenarán para formar una *sola* cadena.

Para continuar una cadena en la siguiente línea, también puede utilizar una barra invertida como último carácter de una línea y, a continuación, pulsar la tecla Intro para comenzar una nueva línea, donde puede seguir escribiendo la cadena.

Ejemplo:

```
"I am a very, very \

    long string"
```

Sin embargo, tenga en cuenta que los espacios iniciales en la segunda línea se evaluarán como parte de la cadena. Por lo tanto, generalmente es preferible utilizar el primer método, es decir, terminar la cadena con " y volver a abrirla con ".

Nombres

Palabras clave en C++

```
asm         do            inline             short        typeid
auto        double        int                signed       typename
bool        dynamic_cast  long               sizeof       union
break       else          mutable            static       unsigned
case        enum          namespace          static_cast  using
catch       explicit      new                struct       virtual
char        extern        operator           switch       void
class       false         private            template     volatile
const       float         protected          this         wchar_t
const_cast  for           public             throw        while
continue    friend        register           true
default     goto          reinterpret_cast   try
delete      if            return             typedef
```

Ejemplos de nombres

Válidos:

```
a         US          us          VOID
_var      SetTextColor
B12       top_of_window
a_very_long_name123467890
```

Inválidos:

```
goto      586_cpu      object-oriented
US$       true         écu
```

Nombres válidos

Dentro de un programa, los *nombres* se utilizan para designar variables y funciones. Las siguientes reglas se aplican al crear nombres, que también se conocen como *identificadores*:

- Un nombre contiene una serie de letras, números o caracteres de subrayado (_). Las diéresis y las letras acentuadas en alemán no son válidas. C++ distingue entre mayúsculas y minúsculas; es decir, las letras mayúsculas y minúsculas son diferentes.
- El primer carácter debe ser una letra o un guión bajo
- No hay restricciones sobre la longitud de un nombre y todos los caracteres del nombre son significativos
- Las palabras clave de C++ están reservadas y no se pueden utilizar como nombres.

La página opuesta muestra palabras clave de C++ y algunos ejemplos de nombres válidos e inválidos. El compilador de C++ utiliza nombres internos que comienzan con uno o dos guiones bajos seguidos de una letra mayúscula. Para evitar confusiones con estos nombres, evite el uso del guión bajo al principio de un nombre.

En circunstancias normales, el enlazador solo evalúa una cantidad determinada de caracteres, por ejemplo, los primeros 8 caracteres de un nombre. Por esta razón, los nombres de objetos globales, como funciones, deben elegirse de modo que los primeros ocho caracteres sean significativos.

Convenciones

En C++, es una práctica estándar utilizar letras minúsculas para los nombres de variables y funciones. Los nombres de algunas variables tienden a estar asociados con un uso específico.

Ejemplos:

```
c, ch               para caracteres
i, j, k, l, m, n    para números enteros, en particular índices
x, y, z             para números de punto flotante
```

Para mejorar la legibilidad de sus programas, debe elegir nombres más largos y más autoexplicativos, como `start_index` o `startIndex` para el primer índice en un rango de valores de índice.

En el caso de proyectos de software, normalmente se aplicarán las convenciones de nombres. Por ejemplo, se pueden asignar prefijos que indiquen el tipo de variable al nombrar las variables.

Variables

Programa de ejemplo

```
// Definición y uso de variables
#include <iostream>
using namespace std;

int gVar1;            // Variables globales,
int gVar2 = 2;        // inicialización explícita

int main()
{
    char ch('A');    // variable local que se está inicializando
                     // or: char ch = 'A';

    cout << "Value of gVar1:    " << gVar1    << endl;
    cout << "Value of gVar2:    " << gVar2    << endl;
    cout << "Character in ch:   " << ch       << endl;

    int sum, number = 3; // Variables locales con
                         // y sin inicialización
    sum = number + 5;
    cout << "Value of sum:      " << sum << endl;

    return 0;
}
```

Sugerencia

Tanto las cadenas como todos los demás valores de tipos fundamentales se pueden imprimir con `cout`. Los números enteros se imprimen en formato decimal de forma predeterminada.

Salida de pantalla

```
Valor de gVar1: }0
Valor de gVar2: 2
Carácter en ch: A
Valor de suma:  8
```

Los datos como números, caracteres o incluso registros completos se almacenan en *variables* para permitir su procesamiento por un programa. Las variables también se conocen como *objetos*, en particular si pertenecen a una clase.

Definición de variables

Una variable debe definirse antes de poder usarla en un programa. Cuando se define una variable, se especifica el tipo y se reserva una cantidad apropiada de memoria. Este espacio de memoria se direcciona por referencia al nombre de la variable. Una definición simple tiene la siguiente sintaxis:

Sintaxis:

```
type name1 [name2 ... ];
```

Esto define los nombres de las variables en la lista `name1 [, name2 ...]` como variables del tipo `type`. Los paréntesis `[ ... ]` en la descripción de la sintaxis indican que esta parte es opcional y puede omitirse. Por lo tanto, se pueden indicar una o más variables dentro de una única definición.

Ejemplos:

```
char c;
int i, counter;
double x, y, size;
```

En un programa, las variables pueden definirse dentro de las funciones del programa o fuera de ellas. Esto tiene el siguiente efecto:

- Una variable definida fuera de cada función es *global*, es decir, puede ser utilizada por todas las funciones
- Una variable definida dentro de una función es *local*, es decir, puede ser utilizada solo en esa función.

Las variables locales normalmente se definen inmediatamente después de la primera llave, por ejemplo al principio de una función. Sin embargo, pueden definirse siempre que se permita una declaración. Esto significa que las variables pueden definirse inmediatamente antes de que el programa las utilice.

Inicialización

Una variable puede inicializarse, es decir, se le puede asignar un valor, durante su definición. La inicialización se logra colocando lo siguiente inmediatamente después del nombre de la variable:

- Un signo igual (=) y un valor inicial para la variable o
- Corchetes que contienen el valor de la variable.

Ejemplos:

```
char c = 'a';
float x(1.875);
```

Cualquier variable *global* que no se inicialice explícitamente tiene un valor predeterminado de cero. Por el contrario, el valor inicial de cualquier variable local que no se inicialice tendrá un valor inicial indefinido.

Palabras clave `const` y `volatile`

Programa de ejemplo

```
// Circunferencia y área de un círculo con radio 2,5

#include <iostream>
using namespace std;

const double pi = 3.141593;

int main()
{
    double area, circuit, radius = 1.5;

    area = pi * radius * radius;
    circuit = 2 * pi * radius;

    cout << "\nTo Evaluate a Circle\n" << endl;

    cout << "Radius:        " << radius    << endl
         << "Circumference: " << circuit   << endl
         << "Area:          " << area      << endl;

    return 0;
}
```

Nota

De manera predeterminada, `cout` genera un número de punto flotante con un máximo de 6 decimales sin ceros finales.

Salida de pantalla

```
Para evaluar un círculo

Radio: 1,5
Circunferencia: 9,42478
Área: 7,06858
```

Se puede modificar un tipo utilizando las palabras claves `const` y `volatil`.

Objetos constantes

La palabra clave `const` se utiliza para crear un objeto de "solo lectura". Como un objeto de este tipo es constante, no se puede modificar en una etapa posterior y se debe inicializar durante su definición.

Ejemplo:

```
const double pi = 3,1415947;
```

Por lo tanto, el programa no puede modificar el valor de `pi`. Incluso una declaración como la siguiente simplemente generará un mensaje de error:

```
pi = pi + 2,0;     // inválido
```

Objetos volátiles

La palabra clave volátil, que rara vez se utiliza, crea variables que pueden ser modificadas no solo por el programa sino también por otros programas y eventos externos. Los eventos pueden iniciarse mediante interrupciones o mediante un reloj de hardware, por ejemplo.

Ejemplo:

```
volatile unsigned long clock_ticks;
```

Incluso si el programa en sí no modifica la variable, el compilador debe asumir que el valor de la variable ha cambiado desde la última vez que se accedió a ella. Por lo tanto, el compilador crea un

código de máquina para leer el valor de la variable cada vez que se accede a ella en lugar de utilizar repetidamente un valor que se ha leído en una etapa anterior.

También es posible combinar las palabras clave `const` y `volatil` al declarar una variable.

Ejemplo:

```
volatile const unsigned time_to_live;
```

En base a esta declaración, la variable `time_to_live` no puede ser modificada por el programa sino por eventos externos.

Ejercicios

1. El operador `sizeof` puede utilizarse para determinar la cantidad de bytes que ocupa en memoria una variable de un tipo determinado. Por ejemplo, `sizeof(short)` es equivalente a 2.

   Escriba un programa en C++ que muestre en pantalla el espacio de memoria requerido por cada tipo fundamental.

2. Escriba un programa en C++ para generar la salida de pantalla que se muestra:

   ```
   I
             "RUSH"
                      \TO\
             AND
   /FRO/
   ```

3. ¿Cuál de las definiciones de variables que se muestran no es válida o no tiene sentido?

   Definición e inicialización de variables:

   ```
   int a(2.5);                  const long large;
   int b = '?';                 char c('\'');
   char z(500);                 unsigned char ch = '\201';
   int big = 40000;             unsigned size(40000);
   double he's(1.2E+5);         float val = 12345.12345;
   ```

4. Escribe un programa en C++ que defina dos variables para números de punto flotante y las inicialice con los valores

   `123,456` y `76,543`

   Luego muestra la suma y la diferencia de estos dos números en la pantalla.

Capítulo 3: Uso de funciones y clases

Este capítulo describe cómo

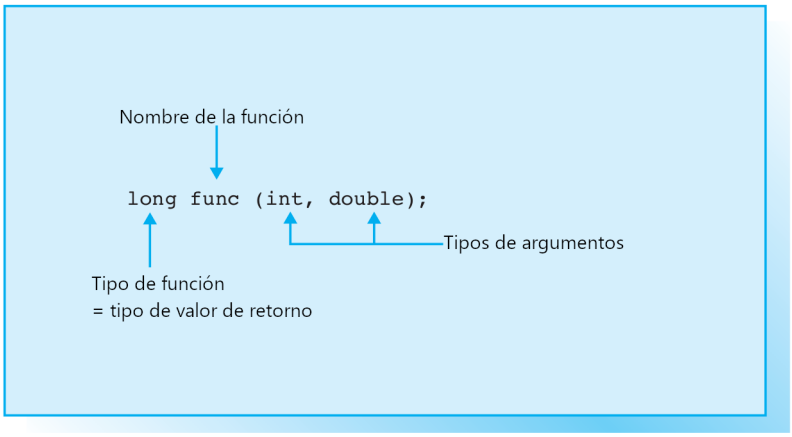
- Declarar y llamar funciones estándar
- Utilizar clases estándar.

Esto incluye el uso de archivos de encabezado estándar. Además, trabajaremos con variables de cadena, es decir, objetos que pertenecen a la clase estándar `string` por primera vez.

Las funciones y clases que defina por su cuenta no se presentarán hasta más adelante en el libro.

Declaración de funciones

Ejemplo de un prototipo de función



El prototipo anterior proporciona la siguiente información al compilador:

- `func` es el nombre de la función
- La función se llama con dos argumentos: el primer argumento es de tipo `int`, el segundo de tipo `double`
- El valor de retorno de la función es de tipo `long`.

Funciones matemáticas estándar

```
double sin (double);           // Seno
double cos (double);           // Coseno
double tan (double);           // Tangente
double atan (double);          // Arco tangente
double cosh (double);          // Coseno Hyperbólico
double sqrt (double);          // Raíz cuadrada/Square Root
double pow (double, double);   // Potencia
double exp (double);           // Función Exponential
double log (double);           // Logaritmo Natural
double log10 (double);         // Logaritmo Base-diez
```

Declaraciones

El compilador debe conocer cada nombre (*identificador*) que aparece en un programa o provocará un mensaje de error. Esto significa que todos los nombres, excepto las palabras clave, deben declararse, es decir, introducirse en el compilador, antes de que se utilicen.

Cada vez que se define una variable o una función, también se declara. Pero, a la inversa, no todas las declaraciones deben ser una definición. Si necesita utilizar una función que ya se ha introducido en una biblioteca, debe declarar la función, pero no necesita redefinirla.

Declaración de funciones

Una función tiene un nombre y un tipo, al igual que una variable. El tipo de la función se define por su *valor de retorno*, es decir, el valor que la función devuelve al programa. Además, el tipo de argumentos requeridos por una función es importante. Cuando se declara una función, el compilador debe recibir información sobre

- El nombre y el tipo de la función
- El tipo de cada argumento.

Esto también se conoce como *prototipo de función*.

Ejemplos:

```
int toupper(int);

double pow(double, double);
```

Esto informa al compilador que la función `toupper()` es de tipo `int`, es decir, su valor de retorno es de tipo `int` y espera un argumento de tipo `int`. La segunda función `pow()` es de tipo `double` y se deben pasar dos argumentos de tipo `double` a la función cuando se la llama. Los tipos de los argumentos pueden ir seguidos de nombres, sin embargo, los nombres se consideran solo como un comentario.

Ejemplos:

```
int toupper(int c);

double pow(double base, double exponent);
```

Desde el punto de vista del compilador, estos prototipos son equivalentes a los prototipos del ejemplo anterior. Ambas uniones son uniones estándar.

Los prototipos de funciones estándar no necesitan ser declarados, ni deberían serlo, ya que ya han sido declarados en los archivos de cabecera estándar. Si el archivo de cabecera se incluye en el código fuente del programa mediante la directiva `#include`, la función puede usarse inmediatamente.

Example:

```
#include <cmath>
```

Siguiendo esta directiva, están disponibles las funciones matemáticas estándar, como `sin()`, `cos()` y `pow()`. Más adelante en este capítulo se pueden encontrar detalles adicionales sobre los archivos de encabezado.

Llamadas a funciones

Programa de ejemplo

```
//  Calculating powers with
//  the standard function pow()

#include <iostream>     // Declaration of cout
#include <cmath>        // Prototype of pow(), thus:
                        // double pow( double, double);
using namespace std;

int main()
{
    double x = 2.5, y;

    // Mediante un prototipo, el compilador genera
    // la llamada correcta o un mensaje de error!

    // Calcula x elevado a la potencia 3:
    y = pow("x", 3.0);   // ¡Error! La cadena no es un número
    y = pow(x + 3.0);    // Error! Solo un argumento
    y = pow(x, 3.0);     // ok!
```

```
    y = pow(x, 3);       // ok! El compilador convierte el
                         // int value 3 to double.

    cout << "2.5 elevado a la potencia 3 da como resultado:        "
         << y << endl;

    // Es posible calcular con pow():
    cout << "2 + (5 elevado a la potencia 2,5) da como resultado: "
         << 2.0 + pow(5.0, x) << endl;

    return 0;
}
```

Salida de pantalla

```
2,5 elevado a la potencia 3 da como resultado: 15,625

2 + (5 elevado a la potencia 2,5) da como resultado: 57,9017
```

Llamadas a funciones

Una *llamada a función* es una expresión del mismo tipo que la función y cuyo valor corresponde al valor de retorno. El valor de retorno se pasa normalmente a una variable adecuada.

Ejemplo:

```
y = pow( x, 3.0);
```

En este ejemplo, primero se llama a la función `pow()` utilizando los argumentos `x` y `3.0`, y el resultado, la potencia x^3, se asigna a `y`.

Como la llamada a la función representa un valor, también son posibles otras operaciones. Por lo tanto, la función `pow()` se puede utilizar para realizar cálculos para valores dobles.

Ejemplo:

```
cout  <<  2.0 + pow( 5.0, x);
```

Esta expresión primero suma el número `2.0` al valor de retorno de `pow(5.0,x)` y luego genera el resultado mediante `cout`.

Se puede pasar cualquier expresión a una función como *argumento*, como una constante o una expresión aritmética. Sin embargo, es importante que los tipos de los argumentos correspondan a los esperados por la función.

El compilador consulta el prototipo para verificar que la función se haya llamado correctamente. Si el tipo del argumento no coincide exactamente con el tipo definido en el prototipo, el compilador realiza una conversión de tipos, si es posible.

Ejemplo:

```
y = pow( x, 3);    // also ok!
```

El valor `3` del tipo `int` se pasa a la función como segundo argumento. Pero como la función espera un valor `double`, el compilador realizará la conversión de tipo de `int` a `double`.

Si se llama a una función con un número incorrecto de argumentos, o si la conversión de tipo resulta imposible, el compilador genera un mensaje de error. Esto le permite reconocer y corregir errores causados por la llamada a funciones en la etapa de desarrollo en lugar de causar errores en tiempo de ejecución.

Ejemplo:

```
float x = pow(3.0 + 4.7);   // Error!
```

El compilador reconoce que el número de argumentos es incorrecto. Además, el compilador emitirá una advertencia, ya que se asigna un `double`, es decir, el valor de retorno de `pow()`, a una variable de tipo `float`.

Tipo `void` para funciones

Programa de ejemplo

```
// Genera tres números aleatorios
#include <iostream>     // Declaración de cin y cout
#include <cstdlib>      // Prototipos de srand(), rand():
                        // void srand( unsigned int seed );
                        // int rand( void );
using namespace std;

int main()
{
    unsigned int seed;
    int z1, z2, z3;

    cout << "--- Random Numbers --- \n" << endl;
    cout << "Para inicializar el generador de números aleatorios, "
         << "\n Por favor, introduzca un valor entero: ";
    cin >> seed;     // Introduzca un número entero

    srand( seed);    // y usarlo como argumento para una
                     // nueva secuencia de números aleatorios.

    z1 = rand();     // Calcular tres números aleatorios.
    z2 = rand();
    z3 = rand();

    cout << "\n:Tres números aleatorios "
         << z1 << "   " << z2 << "   " << z3 << endl;

    return 0;
}
```

Nota

La instrucción `cin >> seed;` lee un entero del teclado, porque `seed` es del tipo `unsigned int`.

Ejemplo de salida de pantalla

```
--- Random Numbers ---

Para inicializar el generador de números aleatorios,
ingrese un valor entero: 7777

Tres números aleatorios: 25435 6908 14579
```

Funciones sin valor de retorno

También puede escribir funciones que realicen una determinada acción pero que no devuelvan un valor a la función que las llamó. El tipo `void` está disponible para funciones de este tipo, que también se conocen como procedimientos en otros lenguajes de programación.

Ejemplo:

```
void srand( unsigned int seed );
```

La función estándar `srand()` inicializa un algoritmo que genera números aleatorios. Dado que la función no devuelve un valor, es de tipo `void`. Se pasa un valor sin signo (`unsigned`) a la función como argumento para generar el generador de números aleatorios. El valor se utiliza para crear una serie de números aleatorios.

Funciones sin argumentos

Si una función no espera un argumento, el prototipo de la función debe declararse como void o las llaves que siguen al nombre de la función deben dejarse vacías.

Ejemplo:

```
int rand( void );     // or     int rand();
```

La función estándar rand() se llama sin ningún argumento y devuelve un número aleatorio entre 0 y 32767. Se puede generar una serie de números aleatorios repitiendo la llamada a la función.

Uso de `srand()` y `rand()`

Los prototipos de funciones para `srand()` y `rand()` se pueden encontrar en los archivos de encabezado `cstdlib` y `stdlib.h`.

Llamar a la función `rand()` sin haber llamado previamente a `srand()` crea la misma secuencia de números que si se hubiera realizado la siguiente declaración:

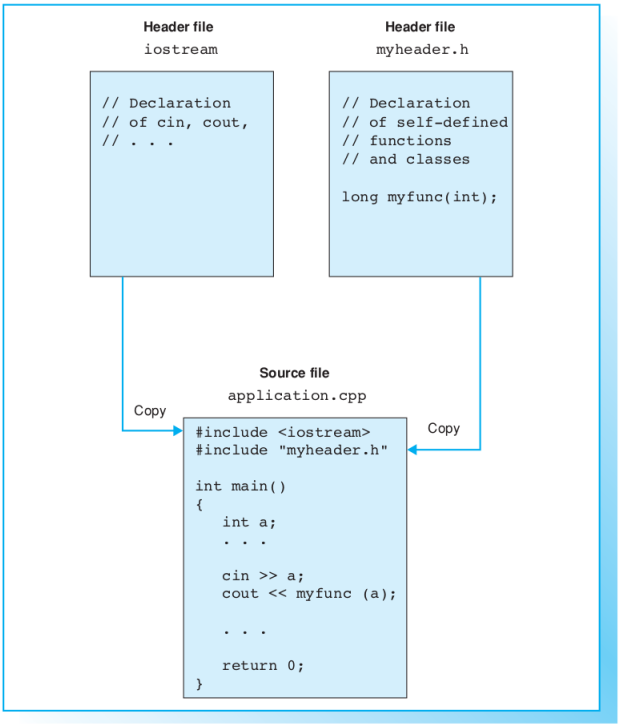
```
srand(1);
```

Si desea evitar generar la misma secuencia de números aleatorios cada vez que se ejecuta el programa, debe llamar a `srand()` con un valor diferente para el argumento cada vez que se ejecuta el programa.

Es común utilizar la hora actual para inicializar un generador de números aleatorios. Consulte el Capítulo 6 para ver un ejemplo de esta técnica.

Archivos de encabezado

Uso de archivos de encabezado



Uso de archivos de encabezado

Los archivos de encabezado son archivos de texto que contienen declaraciones y macros. Al usar una directiva `#include`, estas declaraciones y macros pueden estar disponibles para cualquier otro archivo fuente, incluso en otros archivos de encabezado.

Preste atención a los siguientes puntos cuando use archivos de encabezado:

- Los archivos de encabezado generalmente deben incluirse al comienzo de un programa antes de cualquier otra declaración
- Solo puede nombrar un archivo de encabezado por `#include` directive
- El nombre del archivo debe estar entre corchetes angulares. < ... > o comillas dobles "... ".

Búsqueda de archivos de encabezado

Los archivos de encabezado que acompañan al compilador tienden a almacenarse en una carpeta propia, normalmente llamada "`include`". Si el nombre del archivo de encabezado está entre corchetes angulares < ... >, es habitual buscar archivos de encabezado solo en la carpeta "`include`".

No se busca en el directorio actual para aumentar la velocidad al buscar archivos de encabezado.

Los programadores de C++ suelen escribir sus propios archivos de encabezado y almacenarlos en la carpeta del proyecto actual. Para permitir que el compilador encuentre estos archivos de encabezado, la directiva `#include` debe indicar el nombre de los archivos de encabezado entre comillas dobles.

Ejemplo:

```
#include "project.h"
```

El compilador también buscará en la carpeta actual. El sufijo de archivo `.h` se utiliza normalmente para los archivos de encabezado definidos por el usuario.

Definiciones de clases estándar

Además de los prototipos de funciones estándar, los archivos de encabezado también contienen definiciones de clases estándar. Cuando se incluye un archivo de encabezado, las clases definidas y cualquier objeto declarado en el archivo están disponibles para el programa.

Ejemplo:

```
#include <iostream>
using namespace std;
```

Siguiendo estas directivas, las clases `istream` y `ostream` se pueden utilizar con los flujos `cin` y `cout`. `cin` es un objeto de la clase `istream` y `cout` un objeto de la clase `ostream`.

Archivos de encabezado estándar

Archivos de encabezado de la biblioteca estándar de C++

```
algorithm      ios           map          stack
bitset         iosfwd        memory       stdexcept
complex        iostream      new          streambuf
dequeue        istream       numeric      string
exception      iterator      ostream      typeinfo
fstream        limits        queue        utility
functional     list          set          valarray
iomanip        locale        sstream      vector
```

Nota

Algunos IDE ponen a disposición los antiguos archivos de encabezado `iostream.h` e `iomanip.h`. Dentro de estos archivos, los identificadores de `iostream` e `iomanip` no están contenidos en el espacio de nombres estándar, sino que se declaran globalmente.

Archivos de encabezado de la biblioteca estándar de C

```
assert.h          limits.h          stdarg.h          time.h
ctype.h           locale.h          stddef.h          wchar.h
errno.h           math.h            stdio.h           wctype.h
float.h           setjmp.h          stdlib.h
iso646.h          signal.h          string.h
```

Los archivos de encabezado de la biblioteca estándar de C++ se muestran en la siguiente imagen. No se indican con la extensión de archivo `.h` y contienen todas las declaraciones en su propio espacio de nombres, `std`. Los espacios de nombres se presentarán en un capítulo posterior. Por ahora, es suficiente saber que no se puede hacer referencia directamente a los identificadores de otros espacios de nombres. Si simplemente estipula la directiva

Example:

```
#include <iostream>
```

El compilador no estaría al tanto de los flujos `cin` y `cout`. Para utilizar los identificadores del espacio de nombres `std` de manera global, debe agregar una directiva `using`.

Ejemplo:

```
#include <iostream>
#include <string>
using namespace std;
```

A continuación, puede utilizar `cin` y `cout` sin ninguna sintaxis adicional. También se ha incluido la cadena del archivo de encabezado. Esto hace que la clase de cadena esté disponible y permite manipulaciones de cadenas sencillas en C++. Las siguientes páginas contienen más detalles sobre este tema.

Archivos de encabezado en el lenguaje de programación C

Los archivos de encabezado estandarizados para el lenguaje de programación C se adoptaron para el estándar C++ y, por lo tanto, la funcionalidad completa de las bibliotecas estándar de C está disponible para los programas C++.

Ejemplo:

```
#include <math.h>
```

Las funciones matemáticas se ponen a disposición mediante esta declaración.

Los identificadores declarados en los archivos de encabezado de C son visibles globalmente. Esto puede provocar conflictos de nombres en programas grandes. Por este motivo, cada archivo de encabezado de C, por ejemplo name.h, está acompañado en C++ por un segundo archivo de encabezado, cname, que declara los mismos identificadores en el espacio de nombres `std`. Incluir el archivo `math.h` es, por tanto, equivalente a

Ejemplo:

```
#include <cmath>
using namespace std;
```

Los archivos `string.h` o `cstring` deben incluirse en programas que utilizan funciones estándar para manipular cadenas de C. Estos archivos de encabezado otorgan acceso a la funcionalidad de la biblioteca de cadenas de C y deben distinguirse del archivo de encabezado de cadena que define la clase de cadena.

Cada compilador ofrece archivos de encabezado adicionales para funcionalidades dependientes de la plataforma. Estos pueden ser bibliotecas de gráficos o interfaces de base de datos.

Uso de clases estándar

Programa de ejemplo que utiliza la clase `string`

```
// To use strings.

#include <iostream>      // Declaración de cin, cout
#include <string>        // Declaración de clase string
using namespace std;

int main()
{
    // Define cuatro cadenas:
    string prompt("What is your name: "),
        name,                    // Una vacía
        line( 40, '-'),          // cadena con 40 '-'
        total = "Hello ";        // ¡Es posible!

    cout << prompt;              // Solicitud de aportaciones.
    getline( cin, name);         // Ingresa un nombre en una línea
```

```
    total = total + name;        // Concatena y asigna cadenas.

    cout << line << endl         // Línea de salida y nombre
         << total << endl;
    cout << " Your name is "     // Longitud de salida
         << name.length() << " characters long!" << endl;
    cout << line << endl;

    return 0;
}
```

Nota

Tanto los operadores + y += para concatenación como los operadores relacionales <, <=, >, >=, == y != están definidos para objetos de la clase `string`. Las cadenas se pueden imprimir con `cout` y el operador <<

La clase `string` se presentará en detalle más adelante.

Ejemplo de salida de pantalla

```
What is your name: Rose Summer
---------------------------------------
Hello Rose Summer
Your name is 11 characters long!
---------------------------------------
```

En la biblioteca estándar de C++ se definen varias clases. Entre ellas se incluyen clases de flujo para entrada y salida, pero también clases para representar cadenas o manejar condiciones de error.

Cada clase es un tipo con ciertas propiedades y capacidades. Como se mencionó anteriormente, las propiedades de una clase se definen por sus *miembros de datos* y las capacidades de la clase se definen por sus `métodos`. Los métodos son funciones que pertenecen a una clase y cooperan con los miembros para realizar ciertas operaciones. Los métodos también se conocen como funciones miembro.

Creación de objetos

Un *objeto* es una variable de un tipo de clase, también conocida como una *instancia* de la clase. Cuando se crea un objeto, se asigna memoria a los miembros de datos y se inicializa con los valores adecuados.

Ejemplo:

```
string s("I am a string");
```

En este ejemplo, el objeto `s`, una instancia de la clase estándar `string` (o simplemente una *cadena*/*string*), se define y se inicializa con la constante de cadena que sigue. Los objetos de la clase `string` administran el espacio de memoria necesario para la cadena.

En general, existen varias formas de inicializar un objeto de una clase. Por lo tanto, una cadena se puede inicializar con un cierto número de caracteres idénticos, como ilustra el ejemplo de la página opuesta.

Métodos de llamada

Todos los métodos definidos como *public* dentro de la clase correspondiente se pueden llamar para un objeto. A diferencia de la llamada a una función global, un método siempre se llama para *un objeto en particular*. El nombre del objeto precede al método y está separado del método por un punto.

Ejemplo:

```
s.length();          // object.method();
```

El método `length()` proporciona la longitud de una cadena, es decir, la cantidad de caracteres que contiene. Esto da como resultado un valor de 13 para la cadena `s` definida anteriormente.

Clases y funciones globales

Existen *funciones definidas globalmente* para algunas clases estándar. Estas funciones realizan ciertas operaciones para los objetos que se pasan como argumentos. La función global `getline()`, por ejemplo, almacena una línea de entrada de teclado en una cadena.

Ejemplo:

```
getline(cin, s);
```

La entrada del teclado finaliza presionando la tecla de retorno para crear un carácter de nueva línea, '\n', que no se almacena en la cadena.

Ejercicios

1. Crear un programa para calcular las raíces cuadradas de los números

   ```
   4          12.25          0.0121
   ```

   e imprimirlos como se muestra a continuación. Luego, lea un número desde el teclado y genere la raíz cuadrada de ese número.

   ```
   Number      Square Root
   4           2
   12.25       3.5
    0.0121     0.11
   ```

   Para calcular la raíz cuadrada, use la función `sqrt()`, que está definida por el siguiente prototipo en el archivo de encabezado `math.h` (o `cmath`):

   ```
   double sqrt( double x);
   ```

   El valor de retorno de la función `sqrt()` es la raíz cuadrada de `x`.

2. ¡El programa siguiente contiene varios **errores**! Corrija los errores y asegúrese de que el programa pueda ejecutarse.

   ```
   // ¡Un programa que contiene errores!
   # include <iostream>, <string>
   # include <stdlib>
   # void srand( seed);

   int main()
   {
       string message "\n¡Aprende de tus errores!";
       cout << message << endl;

       int len = length( message);
       cout << "Longitud de la cadena: " << len << endl;

       // Y un número aleatorio además:
       int a, b;
       a = srand(12.5);
       b = rand( a );
       cout << "\nRandom number: " << b << endl;

       return 0;
   }
   ```

3. Cree un programa en C++ que defina una cadena que contenga la siguiente secuencia de caracteres:

   ```
   I have learned something new again!
   ```

   y muestra la longitud de la cadena en la pantalla.

   Lee dos líneas de texto desde el teclado. Concatena las cadenas usando "*" para separar las dos partes de la cadena. Imprime la nueva cadena en la pantalla.

Referencias

Kirch-Prinz U, Prinz P *A Complete Guide to Programming in C++* Jones and Bartlett Publishers, 2002