(75.41) Algoritmos y Programación II

Cátedra Lic. Andrés Juárez

2do cuatrimestre de 2018

Introducción a C++

Carolina Pistillo

Índice

El siguiente apunte es una mera introducción al lenguaje de programación C++. Está elaborado de forma didáctica para introducir un lenguaje que para la mayoria de los casos resulta nuevo. No se pretenden conocimientos previos de ningun lenguaje en específico sino únicamente lo aprendido en Algoritmos y Programación I.

En otras palabras, no se espera que el alumno se aprenda al pie de la letra los fundamentos teóricos dados en este texto sino que logre formarse una idea sobre las nociones básicas de C++ para luego aplicarlo de forma práctica.

1. Lenguajes compilados y C++

1.1. ¿Por qué usar un lenguaje como C++?

En esencia, una computadora es solo un procesador con algo de memoria, capaz de ejecutar pequeñas instrucciones como "almacenar 5 en la ubicación de memoria 23459". ¿Por qué expresaríamos un programa como un archivo de texto en un lenguaje de programación, en lugar de escribir instrucciones del procesador? Las ventajas:

1. **Concisión:** los lenguajes de programación nos permiten expresar secuencias comunes de comandos de forma más concisa. C++ proporciona algunos shorthands especialmente poderosos.

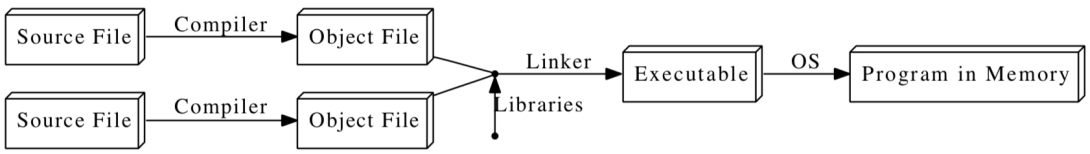
2. **Mantenibilidad:** la modificación del código es más fácil cuando solo requiere unas pocas ediciones de texto, en lugar de reorganizar cientos de instrucciones del procesador. C++ está *orientado a objetos*, lo que mejora aún más el mantenimiento.

3. **Portabilidad:** diferentes procesadores implican diferentes instrucciones disponibles. Los programas escritos como texto pueden traducirse en instrucciones para muchos procesadores diferentes; uno de los puntos fuertes de C++ es que puede usarse para escribir programas para casi cualquier procesador.

C++ es un lenguaje de *alto nivel*: cuando se escribe un programa, las palabras cortas son lo suficientemente expresivas para que no tengas que preocuparte por los detalles de las instrucciones del procesador. C++ da acceso a algunas funcionalidades de menor nivel que otros lenguajes (por ejemplo, direcciones de memoria).

1.2. El proceso de compilación

Un programa va desde archivos de texto o archivos fuente (*source files*) a las instrucciones del procesador de la siguiente manera:

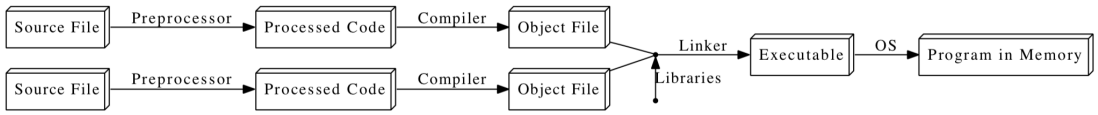


Los archivos objeto (*object file*) son archivos intermedios que representan una copia incompleta del programa: cada archivo fuente solo expresa una parte del programa, por lo que cuando se compila en un archivo objeto, el archivo objeto tiene algunos marcadores que indican de qué piezas faltantes depende. El enlazador (*linker*) toma esos archivos objeto y las bibliotecas compiladas de código predefinido, rellena todos los espacios y genera el programa final, que luego puede ejecutar el sistema operativo (*Operating System*).

El compilador y el enlazador son solo programas regulares. El paso en el proceso de compilación en el que el compilador lee el archivo se llama *parseo*.

En C++, todos estos pasos se realizan con anticipación, antes de comenzar a ejecutar un programa. En algunos idiomas, se realizan durante el proceso de ejecución, lo que lleva tiempo. Esta es una de las razones por las cuales el código en C++ se ejecuta mucho más rápido que el código en muchos lenguajes más recientes.

C++ en realidad agrega un paso adicional al proceso de compilación: el código se ejecuta a través de un preprocesador, que aplica algunas modificaciones al código fuente, antes de ser enviado al compilador. Por lo tanto, el diagrama modificado es:



1.3. Notas generales sobre C++

C++ es inmensamente popular, especialmente para aplicaciones que requieren velocidad y/o acceso a algunas funciones de bajo nivel. Fue creado en 1979 por Bjarne Stroustrup, al principio como un conjunto de extensiones para el lenguaje de programación C. C++ extiende C.

Aunque uno puede escribir programas gráficos en C++, en este curso nos enforcaremos en los programas de consola.

Todo en C++ distingue entre mayúsculas y minúsculas (*case sensitive*): `algunNombre` no es lo mismo que `AlgunNombre`.

2. Hola Mundo

Siguiendo la tradición, utilizaremos el programa "¡Hola, mundo!" como un punto de entrada a las funciones básicas de C++.

2.1. El código

```
1  //Programa de Hola Mundo
2
3  #include <iostream>
4
5  int main()
6  {
7      std::cout<<"¡Hola, mundo!\n";
8      return 0;
9  }
```

2.2. Tokens

Los tokens son la porción mínima del programa que tiene significado para el compilador: los símbolos más pequeños y significativos en el lenguaje. Nuestro código muestra los 6 tipos de tokens, aunque el uso habitual de los operadores no está presente aquí:

Tipo de token	**Descripción/Propósito**	**Ejemplos**
Palabras clave	Palabras con significado especial para el compildor	`int, double, for, auto`
Identificadores	Nombres de cosas que no están integradas en el lenguaje	`cout, std, x, miFuncion`
Literales	Valores constantes básicos cuyo valor se especifica directamente en el código fuente	`"¡Hola, mundo!", 24.3, 0, 'c'`
Operadores	Operaciones matemáticas o lógicas	`+, -, &&, %, <<`
Puntuación/Separadores	Puntuación que define la estructura de un programa	`{} () , ;`
Espacios en blanco	Espacios de varios tipos; ignorados por el compilador	Espacios, tabulaciones, saltos de línea, comentarios

2.3. Explicación línea por línea

1. `//` indica que todo lo que sigue hasta el final de la línea es un comentario: el compilador lo ignora. Otra forma de escribir un comentario es ponerlo entre `/*` y `*/` (por ejemplo, `x = 1 + /* comentario */ 1;`). Un comentario de este estilo puede abarcar varias líneas. Los comentarios existen para explicar cosas no obvias que ocurren en el código.

2. Las líneas que comienzan con `#` son comandos de preprocesador, que generalmente cambian el código que se está compilando. `#include` le dice al preprocesador que descargue el contenido de otro archivo, en este caso el archivo `iostream`, que define los procedimientos para la entrada/salida (*input/output*).

3. `int main() {...}` define el código que se debe ejecutar cuando se inicia el programa. Las llaves representan la agrupación de múltiples comandos en un bloque.

4. - **cout**<< : Esta es la sintaxis para enviar un fragmento de texto a la pantalla.
   - **Espacios de nombres:** En C++, los identificadores se pueden definir dentro de un contexto, una especie de directorio de nombres, llamado espacio de nombres. Cuando queremos acceder a un identificador definido en un espacio de nombres, le decimos al compilador que lo busque en ese espacio de nombres usando el *operador de resolución de alcance* (`::`). En este caso, le estamos diciendo al compilador que busque `cout` en el espacio de nombres `std`, en el que se definen muchos identificadores estándar de C++.

     Una alternativa más prolija es agregar la siguiente línea debajo de la línea 3:

     ```
     4  using namespace std;
     ```

     Esta línea le dice al compilador que debería buscar en el espacio de nombres estándar cualquier identificador que no hayamos definido. Al hacer esto, podemos omitir el prefijo `std::` al usar `cout`. Esta es la práctica recomendada.
   - **Strings**: una secuencia de caracteres como `Hola, mundo` se conoce como un string. Un string que se especifica explícitamente en un programa es un *string literal*.
   - **Secuencias de escape**: `\n` indica un carácter de nueva línea. Es un ejemplo de una secuencia de escape: un símbolo utilizado para representar un carácter especial en un literal de texto. Aquí están todas las secuencias de escape de C++ que puedes incluir en cadenas:

Secuencia de escape	Caracter representado
`\a`	Campana del sistema (pitido)
`\b`	Retroceso
`\f`	Formfeed (salto de página)
`\n`	New line (salto de línea)
`\n`	"Retorno de carro"(devuelve el cursor al inicio de la línea)
`\t`	Tabulación
`\\`	Barra invertida
`\'`	Caracter de comilla simple
`\"`	Caracter de comilla doble
`\algun entero x`	El caracter representado por x

5. `return 0` indica que el programa debe indicar al sistema operativo que se ha completado con éxito.

Tenga en cuenta que cada instrucción finaliza con un punto y coma (excepto los comandos y bloques del preprocesador usando {}). Olvidar estos puntos y comas es un error común para los nuevos programadores de C++.

3. Características básicas del lenguaje

Hasta ahora, nuestro programa no hace mucho. Vamos a modificarlo de varias maneras para demostrar algunas construcciones más interesantes.

3.1. Valores y declaraciones

- Una *declaración* es una unidad de código que hace algo: un componente básico de un programa.
- Una *expresión* es una declaración que tiene un *valor*, por ejemplo, un número, una cadena, la suma de dos números, etc. `4 + 2, x - 1` y "`¡Hola, mundo!\n`" son expresiones.

3.2. Operadores

Podemos realizar cálculos aritméticos con operadores. Los operadores actúan sobre expresiones para formar una nueva expresión. Por ejemplo, podríamos reemplazar "`¡Hola, mundo!\n`" con `(4+2)/3`, lo que haría que el programa imprima el número 2. En este caso, el operador `+` actúa sobre las expresiones `4` y `2`. Tipos de operador:

- **Matemáticos**: `+`, `-`, `*`, `/` y paréntesis tienen sus significados matemáticos habituales, incluido el uso de `-` para la negación. `%` (el operador de módulo) toma el resto de dos números: `6%5` devuelve 1.
- **Lógicos**: se usan para "y"(`&&`), "o"(`||`), y así sucesivamente.
- **A nivel de bits**: se usan para manipular las representaciones binarias de los números. No nos enfocaremos en estos.

3.3. Tipos de datos

Cada expresión es de un tipo. Por ejemplo, `0` es un número entero, `3.142` es un número de *coma flotante* (decimal) y "`¡Hola, mundo!\n`" es un *string* (secuencia de caracteres). Los diferentes tipos de datos toman diferentes cantidades de memoria para su almacenamiento. Aquí están los tipos de datos incorporados que usaremos más a menudo:

Tipo de dato	**Descripción**	**Tamaño**	**Rango**
`char`	Carácter de texto único o entero pequeño. Indicado con comillas simples `('a', '3')`	1 byte	con signo: de -128 a 127 sin signo: de 0 to 255
`int`	Entero más grande	4 bytes	con signo: de -2147483648 a 2147483647 sin signo: de 0 a 4294967295
`bool`	Booleano (verdadero / falso). Indicado con las palabras clave `true` y `false`.	1 byte	Solo `true` (1) y `false` (0).
`double`	Número de punto flotante "doblemente"preciso.	8 bytes	+/- 1.7 e +/- 308 (15 dígitos)

Notas sobre esta tabla:

- Un entero con *signo* es aquel que puede representar un número negativo; un entero *sin signo* nunca se interpretará como negativo, por lo que puede representar un rango más amplio de números positivos. La mayoría de los compiladores asumen con signo si no se especifica.
- En realidad, hay 3 tipos enteros: `short`, `int` y `large`, en orden de tamaño no decreciente (`int` es generalmente un sinónimo de uno de los otros dos). Por lo general, no hay que preocuparse por qué tipo se usa, a menos que esté preocupado por el uso de la memoria o porque esté usando números realmente grandes. Lo mismo ocurre con los 3 tipos de punto flotante, `float`, `double` y `long double`, que están en orden de precisión no decreciente (generalmente hay cierta imprecisión al representar números reales en una computadora).
- Los tamaños/rangos para cada tipo no están completamente estandarizados; los que se muestran arriba son los que se usan en la mayoría de las computadoras de 32 bits.

Una operación solo puede realizarse con tipos compatibles. Puede sumar 34 y 3, pero no puede tomar el resto de un número entero y uno flotante.

Un operador también produce normalmente un valor del mismo tipo que sus operandos; por lo tanto, `1/4` devuelve `0` porque con dos operandos enteros, `/` trunca el resultado a un entero. Para obtener `0.25`, necesitarías escribir algo como `1/4.0` .

Una cadena de texto, por las razones que aprenderemos más adelante, tiene el tipo `char*`.

4. Variables

Es posible que deseemos darle un nombre a un valor para que podamos consultarlo más adelante. Hacemos esto usando variables. Una variable es una ubicación con nombre en la memoria. Por ejemplo, supongamos que queremos usar el valor `4+2` varias veces. Podríamos llamarlo `x` y usarlo de la siguiente manera:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x;
7      x = 4 + 2;
8      cout << x/3 << ' ' << x * 2;
9
10     return 0;
11 }
```

(Tenga en cuenta cómo podemos imprimir una secuencia de valores “encadenando” el símbolo <<).

El nombre de una variable es un token identificador. Los identificadores pueden contener números, letras y guiones bajos (_), y no pueden comenzar con un número.

La línea 6 es la *declaración* de la variable `x`. Debemos decirle al compilador qué tipo será para que sepa cuánta memoria reservar y qué tipo de operaciones se pueden realizar en él.

La línea 7 es la *inicialización* de `x`, donde especificamos un valor inicial para ella. Esto introduce un nuevo operador: `=`, el operador de asignación. También podemos cambiar el valor de `x` más tarde en el código usando este operador.

Podríamos reemplazar las líneas 6 y 7 con una sola declaración que haga tanto la declaración como la inicialización:

```
int x = 4 + 2;
```

Esta forma de declaración/inicialización es más prolija, por lo que es preferible.

5. Input

Ahora que sabemos cómo dar nombres a los valores, podemos hacer que el usuario de los valores de entrada del programa. Esto se demuestra en la línea 7 a continuación:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x;
7      cin >> x;
8
9      cout << x/3 << ' ' << x * 2;
10
11     return 0;
12 }
```

Así como `cout<<` es la sintaxis para generar valores, `cin>>` (línea 7) es la sintaxis para ingresar valores.

Truco mnemotécnico: si tiene problemas para recordar en qué dirección van los paréntesis angulares para `cout` y `cin`, considérelos como flechas que apuntan en la dirección del flujo de datos. `cin` representa la terminal, con datos que fluyen de ella a sus variables; `cout` también representa la terminal y los datos fluyen hacia ella.

Referencias

- Introduction to C++ - MIT Open Course
- C++ Cómo programar - Deitel
- C++ Programming Language - Bjarne Stroustrup