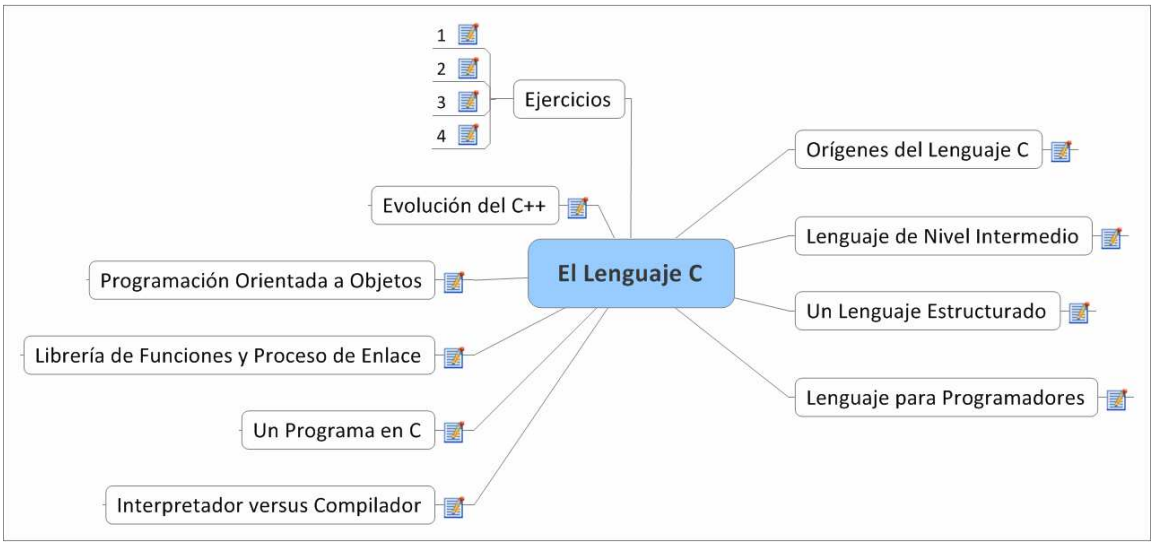
EL LENGUAJE C



Orígenes del Lenguaje C

Dennis Ritchie inventó e implementó por primera vez el lenguaje C en una DEC PDP-11 que usaba sistema operativo UNIX. El lenguaje es el resultado de un proceso de desarrollo que se inició con un antiguo lenguaje llamado BCPL. Martin Richards desarrolló el BCPL, que influenció el invento por parte de Ken Thompson del lenguaje B, el cual a su vez guió el desarrollo del C en la década de los setenta.

Por muchos años, el estándar de facto del lenguaje C, fue la versión suministrada con el sistema operativo UNIX, y fue por primera vez descrito por Brian Kernighan y Dennis Ritchie en su libro *The C Programming Language*, publicado en 1978. En el verano de 1983, se estableció un comité para crear el estándar ANSI (*American National Standards Institute*) que define al lenguaje C. El proceso de estandarización tomo seis años.

El estándar ANSI C fue adoptado finalmente en Diciembre de 1989. Este estándar fue también adoptado por la ISO (*International Standars Organization*), siendo el estándar resultante conocido como ANSI/ISO Standard C, o simplemente ANSI/ISO C. En 1995 se la añadió una enmienda, que entre otras cosas, agregó librerías de funciones adicionales. El estándar C de 1989, junto con la enmienda se usó como documento base para la definición del estándar C++. La versión de C definida en 1989 es referida comúnmente como C89. En 1999 se definió un nuevo estándar para el lenguaje C conocido como C99, que agrega algunas características al estándar C89. Sin embargo, pocos compiladores de C soportan la versión C99.

Lenguaje de Nivel Intermedio

Frecuentemente se dice que el lenguaje C es de nivel intermedio. Eso no significa que sea menos potente, difícil de usar o menos desarrollado que un lenguaje de alto nivel como el Pascal. Tampoco significa que sea similar o presente los inconvenientes asociados al lenguaje ensamblador. La definición del C como lenguaje de nivel intermedio significa que este combina elementos de los lenguajes de alto nivel con la funcionalidad del lenguaje de máquina.

Al ser un lenguaje de nivel intermedio, el C permite la manipulación de bits, bytes y direcciones, los elementos básicos de las funciones computacionales, y a pesar de ello, ser sumamente portable. Portabilidad significa que es posible adaptar un programa escrito de una plataforma a otra.

Todos los lenguajes de alto nivel soporte el concepto de tipos de datos. Un tipo de dato define un conjunto de valores que una variable puede almacenar, junto con un conjunto de operaciones que pueden ser ejecutados utilizando dicha variable. Los tipos de datos más comunes son entero, caracter y real. Aunque el C tiene varios tipos de datos predefinidos, no es un lenguaje "fuertemente tipeado" como el Pascal. De hecho, el C permite casi cualquier tipo de conversión; las variables de tipo entera y caracter pueden ser libremente entremezclados en la mayoría de las expresiones. Tradicionalmente, el lenguaje C no realiza ningún tipo de verificación de error tales como rangos de vectores o compatibilidad de argumentos. Dichas verificaciones son responsabilidad del programador.

Una característica especial del lenguaje C es que permite la manipulación directa de bits, bytes, palabras y apuntadores. Esto lo hace apropiado para la programación a nivel de sistema, donde esas operaciones son comunes. Otro aspecto importante del C es que tiene sólo 32 comandos; esa es una cantidad notablemente inferior a la de cualquier otro lenguaje.

Un Lenguaje Estructurado

La característica que distingue a un lenguaje estructurado es la compartimentalización del código y los datos. La compartimentalización es la habilidad de un lenguaje para seccionar y esconder del resto del programa, toda la información y las instrucciones necesarias para llevar a cabo una tarea específica. Una manera de lograr la compartimentalización es usar subrutinas que emplean variables (temporales) locales. Al usar variables locales, el programador puede escribir subrutinas de tal forma que los eventos que ocurran dentro de ellas, no tengan efectos secundarios en otras partes del programa. Esta capacidad hace que, para programas escritos en lenguaje C, sea fácil compartir secciones de código. Cuando se hace uso de compartimentalización, se requiere saber qué hace una función, pero no cómo lo hace.

> NOTA. El concepto de compartimentalización es ampliado en C++. En C++, una parte del programa puede estrictamente controlar que otras partes del programa pueden ser manipuladas de una porción en particular

Un lenguaje estructurado permite una amplia variedad de posibilidades de programación. Directamente soporta construcciones iterativas tales como *while*, do-*while* y *for*. En un lenguaje estructurado el uso de *goto* es prohibido o desalentado. Los lenguajes estructurados son más recientes que los no estructurados. En la actualidad, muy pocos programadores consideran seriamente el uso de lenguajes no estructurados.

NOTA. Las versiones más recientes de antiguos lenguajes han hecho un intento por incorporar elementos estructurados; el BASIC es un ejemplo. Sin embargo, las desventajas constructivas de estos lenguajes no pueden ser totalmente mitigadas, ya que no fueron diseñados con características estructuradas desde el principio.

El principal componente estructural del lenguaje C es la función. En C, las funciones son bloques constructivos, donde ocurre toda la actividad del programa. Ellas permiten que las diferentes tareas de un programa, sean definidas y codificadas por separado, permitiendo que la construcción sea modular. Cada función debe ejecutar correctamente su función, sin ocasionar efectos secundarios a otras partes del programa. El hecho de crear funciones auto suficientes es extremadamente crítico en grandes proyectos, donde el código de un programador no debe afectar accidentalmente el de otro.

Otra manera de estructurar y compartimentalizar el código en C, es usar bloques de código. Un bloque de código es un grupo de sentencias que son tratadas como una unidad. En C, un bloque de código es creado colocando una grupo de sentencias entre llaves.

```
if (x < 10)
{
   printf("demasiado bajo, inténtelo de nuevo");
   reinicializar_contador(-1);
}
```

En el ejemplo anterior, las sentencias ubicadas entre las llaves se ejecutarán constituyen un bloque de código (unidad lógica) que se ejecutará (en su totalidad) sólo si la expresión condicional que acompaña a la sentencia *if* adopta un valor verdadero. Los bloques de código no sólo permiten que muchos algoritmos sean implementados con claridad, elegancia y eficiencia, sino que además ayudan al programador a conceptualizar la verdadera naturaleza de la rutina.

Lenguaje para Programadores

Cualquiera podría responder a la pregunta ¿es C un lenguaje para programadores? con la pregunta ¿no todos los lenguajes de programación son para programadores? La respuesta a esta última pregunta es No. Lenguajes como el COBOL y el BASIC fueron diseñados para que no programadores pudieran leer, entender y resolver problemas sencillos programando un computador.

Por el contrario, el lenguaje C fue creado, influenciado y verificado en campo por reales programadores. El resultado final es que el C le da a los programadores lo que estos quieren: pocas restricciones, estructuras, funciones independientes y conjunto compacto de comandos. Es realmente impresionante el hecho de que usando lenguaje C, un programador puede obtener casi la misma eficiencia que empleando lenguaje ensamblador, con la modularidad y estructura de un lenguaje de alto nivel. Por esta razón, C es el más popular de los lenguajes de programación.

El hecho de que el C puede ser frecuentemente usado en lugar del lenguaje de ensamblador, contribuye de manera importante a su éxito. El lenguaje ensamblador usa una representación simbólica del código binario que ejecuta el computador. Aunque el lenguaje ensamblador le da al programador el potencial de acometer tareas con la mayor flexibilidad y eficiencia posibles, es notoriamente difícil de usar para desarrollar y depurar un programa. Además, por no ser estructurado, el código en lenguaje ensamblador tiende a ser "código espagueti": una mezcla de saltos, invocaciones e índices. Esta falta de estructura, hace que los programas en lenguaje ensamblador sean difíciles de leer, mejorar y mantener. Quizás más importante aún es el hecho de que las rutinas en lenguaje ensamblador no son portables entre computadores con microprocesadores diferentes.

El lenguaje C fue inicialmente empleado para programación de sistemas. Un programa de sistema es un tipo de programa que forma parte de los sistemas operativos o de sus herramientas de soporte. En la medida en que crece la popularidad del C, muchos programadores han comenzado a usarlo para todo tipo de tareas, debido a su portabilidad, eficiencia y belleza.

Con el advenimiento del C++, algunos pensaron que el C "moriría". Este no ha sido el caso. En primer lugar, no todos los programas requieren de la aplicación de la programación orientada a objetos que proporciona el C++. En segundo lugar, aún existe un cantidad substancial de código escrito en C, que continuará siendo mejorado y mantenido. Aunque el legado más importante del C ha sido la fundación del C++, continuará siendo un lenguaje de amplio uso, por muchos años.

Interpretador versus Compilador

Es importante entender que son los lenguajes de programación los que definen la naturaleza de los programas y no la manera en que sean ejecutados. Existen dos métodos generales para ejecutar programas: pueden ser compilados o interpretados. Aunque en general los diferentes lenguajes de programación pueden ser interpretados o compilados, algunos fueron diseñados para ser interpretados y otros para ser compilados. Por ejemplo, el JAVA fue diseñado

para ser interpretado, mientras que el C lo fue para ser compilado. Adicionalmente, el C fue específicamente optimizado como lenguaje compilado.

En su forma más sencilla, un interpretador lee el código fuente del programa una línea por vez, ejecutando las instrucciones contenidas en dicha línea. Esa es la manera en que trabajaban las primeras versiones de BASIC: En lenguajes como el JAVA, el código es primero convertido a un código intermedio, que posteriormente es interpretado. En cualquier caso, se requiere de un interpretador para ejecutar el programa.

Un compilador lee el programa completo y lo convierte en código objeto, el cual no es más que un traducción del código fuente a un formato que pueda ser ejecutado directamente por el microprocesador. El código objeto es también llamado código binario o código de máquina. Una vez que el programa es compilado, ya no son necesarias las líneas de código fuente para la ejecución del programa.

En general un programa interpretado se ejecuta más lentamente que un programa compilado.

Un Programa en C

La tabla a continuación muestra los 32 comandos básicos, que en conjunto con la sintaxis formal del C, constituyen el estándar C89. Todos los comandos del C son escritos en minúsculas. En C, una letra en minúscula es diferente de la misma letra en mayúscula. El nombre de un comando no puede ser usado para un propósito diferente del originalmente previsto; por ejemplo, no puede ser usado como nombre de función o de variable.

Tabla 1.1 - Comandos Básicos del Lenguaje C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Todos los programas en C consisten de una o más funciones. La única función indispensable es llamada *main*(), y es la primera función invocada cuando se ejecute el programa. En un código C "bien escrito", la función *main*() esboza lo que hace el programa. Este "esbozo" está compuesto de invocaciones a funciones. Aun main() no es un comando, debe ser tratado como si lo fuera.

Librería de Funciones y Proceso de Enlace

Estrictamente hablando, es posible crear un programa útil y funcional en C, con sólo sentencias creadas por el programador. Sin embargo esto es raro, ya que el C no contiene comandos para ejecutar acciones tales como operaciones de entrada y salida, cálculos matemáticos de alto nivel o manejo de cadenas. Como resultado, la mayoría de los programas incluyen llamadas a diversas funciones contenidas en las librerías estándar del C.

El lenguaje C define una librería estándar que proporciona funciones para llevar a cabo las tareas más comúnmente requeridas. Cuando se usa una función que no es parte del programa escrito, el compilador "recuerda" su nombre. Posteriormente el enlazador (*linker*) combina el código escrito por el programador con el existente en la librería estándar. Este proceso es llamado enlace.

Las funciones contenidas en la librería lo están en formato relocalizable. Esto significa, que no está definidas de manera absoluta, las direcciones de memoria de las instrucciones que componen dichas funciones. Cuan se enlaza el programa con la librería, se crean las direcciones de memoria definitivas.

Programación Orientada a Objetos

Aunque la programación estructurada alcanzó un enorme éxito como estrategia para resolver complejos problemas de programación, a finales de los ochenta, algunas de las deficiencias de la programación estructurada se hicieron demasiado evidentes para ser ignoradas. En primer lugar, es natural pensar en los datos y lo que con ellos se puede hacer, como ideas relacionadas. En segundo lugar, los programadores se encontraron constantemente reinventando soluciones para viejos problemas, concepto opuesto al término reusabilidad. La idea detrás de la reusabilidad es fabricar componentes con características específicas y conocidas, que puedan ser insertados en los programas cuando sea necesario. Este modelo proviene del mundo de la electrónica; cuando un ingeniero requiere un transistor no, lo inventa, sino que busca entre los existentes el que más le conviene. No existía alternativa similar en el mundo de la ingeniería de software.

Por otra parte, la manera en que se usan hoy en día los computadores (con menús, botones y ventanas), ha fomentado el empleo de un estilo de programación orientado a eventos. Orientado a eventos significa que cuando ocurra un evento (el usuario presiona un botón o selecciona un menú), el programa debe responder de manera acorde. Los programas antiguos obligaban al usuario a navegar a través de una serie de pantallas. Los

programas modernos, orientados a eventos, presentan todas las opciones de una sola vez, y responden a las acciones del usuario.

La programación orientada a objetos intenta responder a esas necesidades, proporcionando técnicas para manejar problemas complejos, permitiendo la reusabilidad de los componentes de software y acoplando la información con las tareas que se usan para manipularla. La esencia de la programación orientada a objetos es tratar la información y los procedimientos que la operan, como si fueran un único objeto, una entidad auto contenida con identidad y características propias.

Evolución del C++

Al comenzar a evidenciarse la importancia del análisis, diseño y programación orientados a objeto, Bjarne Stroustrup tomó el lenguaje para aplicaciones comerciales más popular (lenguaje C) y lo extendió de tal manera que proporcionara las características que facilitaran la programación orientada a objetos. Él creó el C++ y en menos de una década pasó de ser un lenguaje utilizado por un grupo de desarrolladores de AT&T a ser la selección de alrededor de un millón de programadores alrededor del mundo. En la actualidad, el C++ es el lenguaje predominante para el desarrollo de aplicaciones comerciales.

Aunque es cierto que el C++ es un súper conjunto del C, y que virtualmente cualquier programa "legalmente" escrito en C, es también "legal" en C++, las diferencias entre C y C++ son muy significativas. El C++ se beneficia de su relación con el C, al facilitar su uso por parte de los programadores de C. Sin embargo, para obtener todo el beneficio del C++, muchos programadores han tenido que "desaprender" muchos conceptos para "reaprenderlos" de una manera completamente nueva y notablemente más eficaz para la solución de problemas.

Ejercicios

1

Analice el siguiente programa y trate de deducir qué hace.

```
#include <iostream.h>
#include <stdlib.h>
int main()
```

```
{
    int x= 5;
    int y= 7;
    cout << "\n";
    cout << x+y << " " << x*y;
    cout << "\n";
    system("PAUSE");
    return 0;
}
```

2

Transcriba, compile y ejecute el programa del ejercicio 1 y describa qué hace. ¿Es lo que usted había pensado?.

3

Transcriba y compile el siguiente programa. ¿Qué error recibe?.

```
#include <stdlib.h>
int main ()
{
    cout << "  ****   **   **\n**   ** ** **\n";
    cout << "**   ** ****\n**   ** ****\n";
    cout << "**   ** ** **\n ****   **   **\n\n";
    system ("PAUSE");
    return 0;
}
```

4

Corrija el error del programa del ejercicio 3 y describa qué hace.