1. Introducción al Lenguaje C

El lenguaje de programación C fue creado por Dennis Ritchie en 1972 en Bell Telephone Laboratories, con el objetivo de reescribir un sistema operativo, el UNIX, en un lenguaje de alto nivel, para poder adaptarlo (*portarlo*) a diferentes arquitecturas. Por este motivo sus creadores se propusieron metas de diseño especiales, tales como:

- **Posibilidad de acceder al "bajo nivel" (poder utilizar todos los recursos del hardware).**
- **Código generado eficiente en memoria y tiempo (programas pequeños y veloces)**
- **Compilador portable (implementable en todas las arquitecturas)**

La primera definición oficial del lenguaje fue dada en 1978 por **Brian Kernighan y Dennis Ritchie** en su libro "El lenguaje de programación C". Este lenguaje fue llamado "C K&R". En 1983 se creó el comité ANSI que en 1988 estableció el standard ANSI C, con algunas reformas sobre el C K&R. Simultáneamente Kernighan y Ritchie publicaron la segunda edición de su libro, describiendo la mayor parte de las características del ANSI C.

m2.192cmm5.395cmm2.192cmm5.03cmm2.192cm *Brian Kernighan* *Dennis Ritchie*

Actualmente existen implementaciones de C para todas las arquitecturas y sistemas operativos, y es el lenguaje más utilizado para la programación de sistemas. Por su gran eficiencia resulta ideal para la programación de sistemas operativos, drivers de dispositivos, herramientas de programación. El 95 % del sistema operativo UNIX está escrito en C, así como la gran mayoría de los modernos sistemas y ambientes operativos y programas de aplicación que corren sobre ellos.
Mas información ...

Características del lenguaje

C es un lenguaje compilado. A nivel sintáctico, presenta grandes similitudes formales con Pascal, pero las diferencias entre ambos son importantes. A pesar de permitir operaciones de bajo nivel, tiene las estructuras de control, y permite la estructuración de datos, propias de los lenguajes procedurales de alto nivel como Pascal.

Un programa en C es, por lo general, más sintético que en otros lenguajes procedurales como Pascal; la idea central que atraviesa todo el lenguaje es la minimalidad. La definición del lenguaje consta de muy pocos elementos; tiene muy pocas palabras reservadas. Como rasgo distintivo, en C no existen, rigurosamente hablando, funciones o procedimientos de uso general del programador. Por ejemplo, no tiene funciones de entrada/salida; la definición del lenguaje apenas alcanza a las estructuras de control y los operadores. La idea de definir un lenguaje sin funciones es, por un lado, hacer posible que el compilador sea pequeño, fácil de escribir e inmediatamente portable; y por otro, permitir que sea el usuario quien defina sus propias funciones cuando el problema de programación a resolver tenga requerimientos especiales.

Sin embargo, se ha establecido un conjunto mínimo de funciones, llamado la **biblioteca standard** del lenguaje C, que todos los compiladores proveen, a veces con agregados. La filosofía de la biblioteca standard es la portabilidad, es decir, casi no incluye funciones que sean específicas de un sistema operativo determinado. Las que incluye están orientadas a la programación de sistemas, y a veces no resultan suficientes para el programador de aplicaciones. No provee, por ejemplo, la capacidad de manejo de archivos indexados, ni funciones de entrada/salida interactiva por consola que sean seguras ("a prueba de usuarios"). Esta deficiencia se remedia utilizando bibliotecas de funciones "de terceras partes" (creadas por el usuario u obtenidas de otros programadores).

El usuario puede escribir sus propios procedimientos (llamados **funciones** aunque no devuelvan valores). Aunque existe la noción de bloque de sentencias, el lenguaje se dice *no* estructurado en bloques porque no pueden definirse funciones dentro de otras. Las funciones de la biblioteca standard no tienen ningún privilegio sobre las del usuario y sus nombres no son palabras reservadas; el usuario puede reemplazarlas por sus propias funciones simplemente dándoles el mismo nombre.

El lenguaje entrega completamente el control de la máquina subyacente al programador, no realizando controles de tiempo de ejecución. Es decir, no verifica condiciones de error comunes como *overflow* de variables, errores de entrada/salida, o consistencia de argumentos en llamadas a funciones. Como resultado, es frecuente que el principiante, y aun el experto, cometan errores de programación que no se hacen evidentes enseguida, ocasionando problemas y costos de desarrollo. Permite una gran libertad sintáctica al programador. No es fuertemente tipado. Cuando es necesario, se realizan conversiones automáticas de tipo en las asignaciones, a veces con efectos laterales inconvenientes si no se tiene precaución. Una función que recibe determinados parámetros formales puede ser invocada con argumentos reales de otro tipo.

Se ha dicho que estas características "liberales" posibilitan la realización de proyectos complejos con más facilidad que otros lenguajes como Pascal o Ada, más estrictos; aunque al mismo tiempo, así resulta más difícil detectar errores de programación en tiempo de compilación. En este sentido, según los partidarios de la tipificación estricta, C no es un buen lenguaje. Gran parte del esfuerzo de desarrollo del standard ANSI se dedicó a dotar al C de elementos para mejorar esta deficiencia.

Los tipos de datos no tienen un tamaño determinado por la definición del lenguaje, sino que diferentes implementaciones pueden adoptar diferentes convenciones. Paradójicamente, esta característica obedece al objetivo de lograr la portabilidad de los programas en C. El programador está obligado a no hacer ninguna suposición sobre los tamaños de los objetos de datos, ya que lo contrario haría al software dependiente de una arquitectura determinada.

Una característica especial del lenguaje C es que el pasaje de argumentos a funciones se realiza siempre por valor. ¿Qué ocurre cuando una función debe modificar datos que le son pasados como argumentos? La única salida es pasarle -por valor- la dirección del dato a modificar. Las consecuencias de este hecho son más fuertes de lo que parece a primera vista, ya que surge la necesidad de todo un conjunto de técnicas de

manejo de punteros que no siempre son bien comprendidas por los programadores poco experimentados, y abre la puerta a sutiles y escurridizos errores de programación. Quizás este punto, junto con el de la ausencia de chequeos en tiempo de ejecución, sean los que le dan al C fama de "difícil de aprender".

Por último, el C no es un lenguaje orientado a objetos, sino que adhiere al paradigma tradicional de programación procedural. No soporta la orientación a objetos propiamente dicha, al no proporcionar herramientas fundamentales, como la herencia. Sin embargo, algunas características del lenguaje permiten que un proyecto de programación se beneficie de todas maneras con la aplicación de algunos principios de la orientación a objetos, tales como el ocultamiento de información y el encapsulamiento de responsabilidades. El lenguaje C++, orientado a objetos, **no** es una versión más avanzada del lenguaje o un compilador de C con más capacidades, sino que se trata de un lenguaje completamente diferente.

Algunas nuevas características de C99 son:

- Matrices de tamaño variable
- Soporte de números complejos
- Tipos **long long int** y **unsigned long long int** de al menos 64 bits
- Familia de funciones **vscanf()**
- Comentarios al estilo de C++ prefijando las líneas con la secuencia "//".
- Familia de funciones **snprintf()**
- Tipo boolean

El ciclo de compilación

Las herramientas esenciales de un ambiente de desarrollo, además de cualquier editor de textos, son el **compilador**, el **linkeditor** o *linker* y el **bibliotecario**. A estas herramientas básicas se agregan otras, útiles para automatizar la compilación de proyectos extensos, almacenar y recuperar versiones de programas fuente, chequear sintaxis en forma previa a la compilación, etc. Según el ambiente operativo y producto de software de que se trate, estas herramientas pueden encontrarse integradas en una interfaz de usuario uniforme, en modo texto o gráfico, o ser comandos de línea independientes, con salidas de texto simples.

Compilador

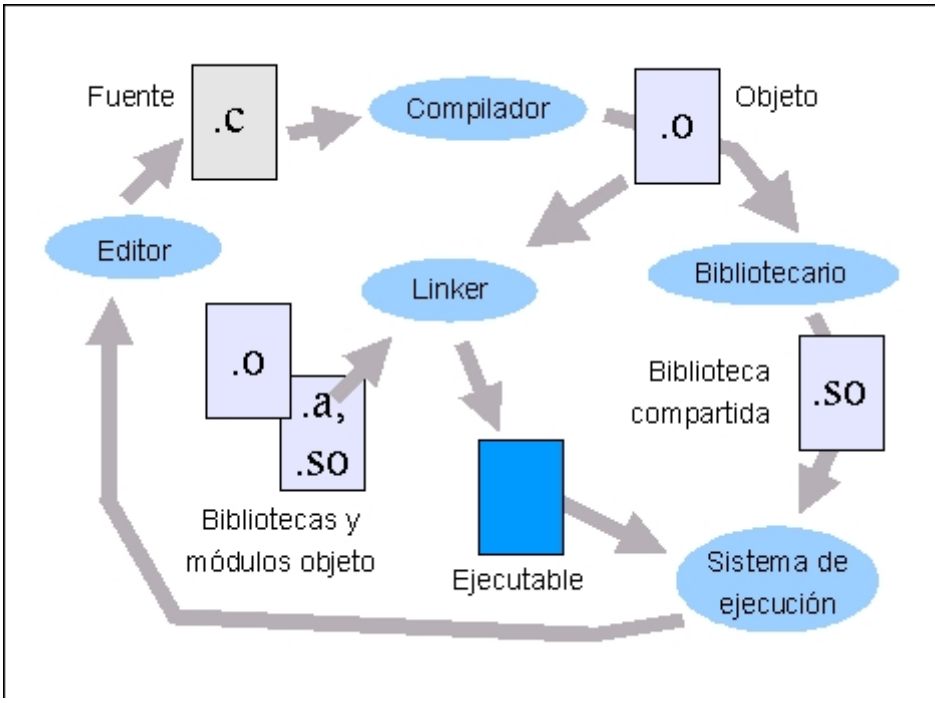
El compilador acepta un archivo fuente, posiblemente relacionado con otros (una **unidad de traducción**), y genera con él un **módulo objeto**. Este módulo objeto contiene porciones de código ejecutable mezclado con referencias, aún no resueltas, a variables o funciones cuya definición no figura en los fuentes de entrada. Estas referencias quedan en forma simbólica en el módulo objeto hasta que se resuelvan en un paso posterior.

Linkeditor o *linker*

El linkeditor recibe como entrada un conjunto de módulos objeto y busca resolver, o enlazar, las referencias simbólicas en ellos, buscando la definición de las variables o funciones faltantes en los mismos objetos o en bibliotecas. Estas pueden ser la biblioteca standard u otras provistas por el usuario. Cuando encuentra la definición de un objeto buscado (es decir, de una variable o función), el linker la copia en el archivo resultante de salida (la *resuelve*). El objetivo del linker es resolver todas las referencias pendientes para producir un programa ejecutable.

Bibliotecario

El bibliotecario es un administrador de módulos objeto. Su función es reunir módulos objeto en archivos llamados bibliotecas, y luego permitir la extracción, borrado, reemplazo y agregado de módulos. El conjunto de módulos en una biblioteca se completa con una tabla de información sobre sus contenidos para que el linker pueda encontrar rápidamente aquellos módulos donde se ha definido una variable o función, y así extraerlos durante el proceso de linkedición. El bibliotecario es utilizado por el usuario cuando desea mantener sus propias bibliotecas. La creación de bibliotecas propias del usuario ahorra tiempo de compilación y permite la distribución de software sin revelar la forma en que se han escrito los fuentes y protegiéndolo de modificaciones.



m17.8cm

El ciclo de compilación produce un ejecutable a partir de archivos fuente

El primer ejemplo

El clásico ejemplo de todas las introducciones al lenguaje C es el programa **hello.c**:

```
#include <stdio.h>
/* El primer programa! */
main()
{
        printf("Hola, gente!\n");
}
```

Ejecute los siguientes pasos para ver el primer ejemplo en C funcionando

- 1.Bajar el archivo hola.c y guardar en el directorio local de trabajo
- 2. Sin cambiarde directorio, ejecutar el comando:

  $ make hola

- 3. Finalmente ejecutar el programa asi:

  de trabajo)

Se puede hacer lo mismo de esta manera:

- 1. Bajar el archivo hola.c y guardar en el directorio local de trabajo
- 2. Sin cambiar de directorio, ejecutar el comando:

  $ gcc hola.c- o hola
- 3. Finalmente ejecutar el programa asi:

  (el . significa el directorio actual de trabajo)

La diferencia es que en el primer caso utilizamos la herramienta make (mas información) que nos asiste en la compilación de proyectos, mientras que en el segundo caso invocamos directamente al compilador. En este último caso, le decimos al compilador (gcc) que compile el archivo hola.c y que la salida (-o de out) se llame hola. Probar que ocurre si hacemos: gcc hola.c.
En el primer caso el utilitario make hace todo esto por nosotros.

Funcionamiento el programa

- Este programa minimal comienza con una **directiva de preprocesador** que indica incluir en la unidad de traducción al archivo *header* **stdio.h**. Este contiene, entre otras cosas, la declaración (o **prototipo**) de la función de salida de caracteres **printf**. Los prototipos se incluyen para advertir al compilador de los tipos de las funciones y de sus argumentos.
- Entre los pares de caracteres especiales **/*** y ***/** se puede insertar cualquier cantidad de líneas de comentarios.
- La función **main()** es el cuerpo principal del programa (es por donde comenzará la ejecución). Terminada la ejecución de main(), terminará el programa.
- La función printf() imprimirá la cadena entre comillas, que es una **constante string** terminada por un carácter de **nueva línea** (la secuencia especial "\n").

Ejercicios

1. ¿Qué nombres son adecuados para los archivos fuente C?
2. Describa las etapas del ciclo de compilación.
3. ¿Cuál sería el resultado de:

- Editar un archivo fuente?
- Ejecutar un archivo fuente?
- Editar un archivo objeto?
- Compilar un archivo objeto?

- Editar una biblioteca?

4. ¿Qué pasaría si un programa C **no** contuviera una función **main()**?. Haga la prueba con hola.c

5. Edite el programa **hola.c** y modifíquelo según las pautas que siguen. Interprete los errores de compilación. Si resulta un programa ejecutable, vea qué hace el programa.

- Quite los paréntesis de main()
- Quite la llave izquierda de main()
- Quite las comillas izquierdas
- Quite los caracteres "\n"
- Agregue al final de la cadena los caracteres "\n\n\n\n"
- Agregue al final de la cadena los caracteres "\nAdiós mundo!\n"
- Quite las comillas derechas
- Quite el signo punto y coma.
- Quite la llave derecha de main()
- Agregue un punto y coma en cualquier lugar del texto
- Agregue una coma o un dígito en cualquier lugar del texto
- Reemplace la palabra **main** por **program**, manteniendo los paréntesis.
- Elimine la apertura o cierre de los comentarios

Ejercicios Adicionales

Ejercicios Avanzados