***Introducción a Lenguaje C (desde la terminal).***

*El puente entre el lenguaje de una computadora (código binario) y el lenguaje humano son los lenguajes de programación; en el caso del lenguaje C, ese intermediario que permite que la computadora pueda entender lo que queramos decirle se llama:* ***compilador*** *(herramienta que nos pasará nuestro código de programación a código de maquina). La extension de los archivos C es:* ***.c****; la extensión determina que estás escribiendo, justamente, un archivo en lenguaje C. Dicho lo anterior, cuando el compilador interpreta a nuestro archivo.c, se encargará que pase dicho archivo por 4 etapas de programación, las cuáles son (muy rapidamente):*

***1. Preprocessor:*** *En esta etapa se interpretan las directivas al preprocesador. Entre otras cosas, las variables inicializadas con #define son sustituidas en el código por su valor en todos los lugares donde aparece su nombre. Es decir, lo que hace es reemplazar la información correspondiente de cada libreria en el mismo espacio donde se ha incluido la libreria en cuestión (para que la computadora reconozca las funciones de ellas dentro del código; pues, debe recordar que, son las librerias las que poseen esas funciones y si no las incluye... )*

***2. Compilador:*** *Se encarga de pasar la información al Assembler. Basicamente "compilar" lo que hace es traducir el texto o código que tenemos, en lenguaje C u otro lenguaje de programación, en un texto semi-legible para el ordenador; de tal manera que lo empieza a interpretar (no es su natural lenguaje ordinario, pero ya es algo más parecido a él). Realmente lo convierte a otro tipo de código fuente llamado* ***código ensamblador.*** *Estas instrucciones son de nivel inferior y están más cerca de las instrucciones binarias que el procesador de una computadora puede entender directamente. Por lo general, operan en bytes en sí mismos, a diferencia de abstracciones como nombres de variables.*

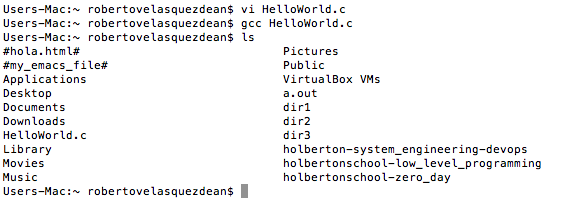
***3. Assembler:*** *El ensamblado transforma el programa, escrito en lenguaje ensamblador, a código objeto: un archivo binario en lenguaje de máquina ejecutable por el procesador, la conocida:* ***“salida del ensamblador”.*** *Ya esto sí sería un texto legible completamente para el ordenador, pues ya sería* ***código binario.***

***4. Linking:*** *se encarga de procesar la información del “C header”, archivo con extensión* ***.h*** *o archivo de cabezera* ***(header files)****: todo lo que tiene que ver con las instrucciones para cargar una libreria, las cuales se cargan por medio de la instrucción “#include”; pero, principalmente, en la vinculación (linking) es donde las versiones compiladas previamente de las bibliotecas que incluimos con anterioridad, en realidad, se combinan con el binario de nuestro programa. Así que terminamos con un archivo binario, “a.out”, que es el código de la máquina combinada con su código fuente (el archivo original tipo .c y sus librerias usadas).*

*Retomando: es preciso incorporar de algún modo el código binario de estas funciones a nuestro ejecutable ya resultante y el proceso “Linking” se encarga de eso. Eso se refiere a que, las variables inicializadas de cada libreria pasan a ser sustituidas en el código directamente por su valor ya en el ejecutable; es decir, es quien interpreta los valores de todas las variables, pero no sólo eso: es quién se encarga de interpretar la información relacionada a definiciones de macros, constantes, declaraciones de funciones, etc y los permite ejecutar.*

***La dinamica y el procedimiento inicial en el compilador es el siguiente:***

*Luego de crear un archivo tipo .c con nombre de, por ejemplo: "HelloWorld.c"* ***(nuestro código fuente, entrada)*** *en nuestro editor Vi, ejecutamos nuestro compilador con dicho archivo, tal que así: "****gcc******HelloWorld.c****". Lo que hará la terminal, seguido a eso, es crearnos un archivo ejecutable llamado:* ***"a.out"******(código maquina, salida)*** *dentro del directorio actual. Hay otro comando que hace exactamente lo mismo:* ***clang*** *(revise con comando ls);*



***"a.out"*** *aquí se muestra como verde claro en la shell bash (a algunos no, como en mi caso), eso le ayuda a entender que su bit ejecutable está configurado, podemos verificarlo ejecutando un comando ls -l. Todos los usuarios tendrán habilitados los permisos de ejecución, cualquiera sea el usuario, tal que así:*

***Entonces, ¿cómo se ejecuta nuestro archivo ejecutable restante?***

***“a.out”,*** *como programa ejecutable, para ser ejecutado sólo es necesario poner, en una línea de código, un punto y una barra (ajustadas en su totalidad) y luego escribir: "a.out" (todo ajustado). Tal que así:* ***"./a.out";***

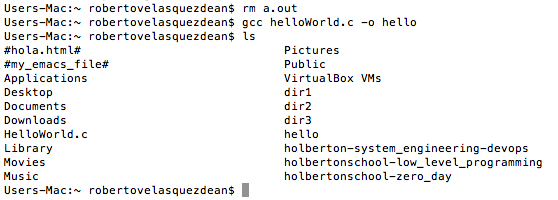


*Si se da cuenta, nuestra terminal, ha podido por fin ejecutar, por medio de su propio lenguaje de maquina (binario) las funciones que habíamos escrito en nuestro archivo .c inicial; esas funciones eran imprimir justamente la palabra: “HelloWorld” (tal como se ve en la imagen).*

*Ahora, otra forma en que podemos hacer esto si quisiéramos que el nombre del archivo ejecutable fuera otro* ***(y no “a.out” porque se ve muy feo)*** *como por ejemplo, darle el nombre de* ***“hello”*** *(pero puede ser cualquiera)... Haga esto: elimine ese ejecutable "a.out" con un comando "rm" (****"rm a.out"****, así debe verse).*

*Y ahora podría decirle al compilador que compile "HelloWorld.c", pero no acaba ahí, usted es quién debe poner en su salida una opcion del tipo "-o" en el archivo "helloWorld.c" para darle un nuevo nombre al archivo .c que sería el ejecutable (el nuevo nombre que le daremos al archivo que por defecto se llama: a.out); sólo justo después de eso (de colocar “-o”, que viene de “output”; es decir, yo decido la salida), hace un espacio y coloca al lado el nombre que quiera darle a su nuevo archivo tipo c ejecutable, habíamos definido que se llamaría "hello"; entonces, su linea de código debería verse algo así:* ***"gcc helloWorld.c -o hello"*** *...*

*Cuando haga esto, si observa su directorio de trabajo, verá un programa "hello" y podrá ejecutarlo tal como hicimos con el primero (a.out).*



*Para ejecutar el archivo ejecutable resultante sería así:* ***"./hello"****... Nos imprimiría, también, un "Hello World"... exactamente como se hizo antes.*

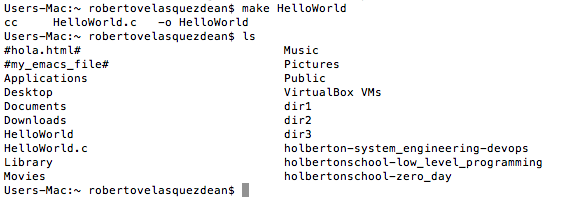


*Ahora, hay una cosa más que podría hacer aquí (en caso que no guste de ninguna de las dos alternativas anteriores). Digamos que no quería escribir esa opción "-o", así que eliminaré "hello" con un comando "rm". La otra solución a la que podemos recurrir, para crear nuestro ejecutable final, es ejecutando un comando del tipo* ***"make". Sin embargo, es necesario aclarar que el nombre que recibirá el archivo ejecutable final será el mismo nombre que le pasamos al archivo c. inicial que creamos para que luego pueda ser compilado; es decir: “HelloWorld.c”***

*Normalmente con una compilación más compleja del tipo* ***"make"*** *tendríamos un archivo ejecutable también, pero "make" es bastante inteligente; por medio de la observación a un archivo fuente (como es el caso de “HelloWorld.c”),* ***make*** *hará una interpolación y entenderá lo que le pidamos que haga sin ni siquiera haber declarado antes que ibamos a hacer un archivo ejecutable a partir de ese archivo inicial c.*

***El comando “make” simplifica todos estos 4 pasos asistiendose del comando clang. (compilador); popularmente se ha aceptado dentro de la jerga de la industria que, para referirnos a los 4 pasos en suma, basta con decir “compilación”, cuando realmente “la compilación” es sólo un paso más de los 4 totales.***

*Si por ejemplo solo ejecuto la siguiente frase, en nuestro caso,* ***"make HelloWorld",*** *lo siguiente que hará “make” es buscar en mi directorio de trabajo actual y ver si hay un archivo fuente que se correlacione con este objetivo que solicite crear (como sí lo hay efectivamente ya que existe el archivo “HelloWorld.c”); entonces, de ser así, ejecutará el compilador apropiado y lo construirá; así que si hago esto, verás que* ***"make"*** *ejecutará automaticamente el compilador y la opción "-o" , todo lo hace por ti. Veamos:*

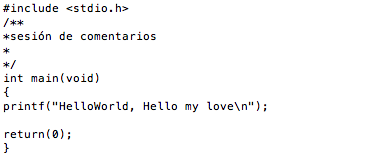


*Así que una vez más mi programa "helloWorld" se compila en el lugar correcto y todo lo que tenía que hacer era escribir "make helloWorld" solamente. Para ejecutar el ejecutable resultante, lo mismo:* ***./****HelloWorld.*



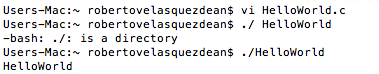
***Algo super importante:*** *En caso tal que decidamos hacer modificaciones sobre el archivo c. inicial, por el motivo que sea, y queremos que el archivo ejecutable resultante nos refleje dichos cambios al ejecutarlo en pantalla; debemos nuevamente, antes de ejecutar el archivo ejecutable, compilar. Para todo cambio que haga sobre su archivo .c original y que luego desee ejecutar en su archivo resultante, debe primero volver a compilar.*

*Si por ejemplo deseamos cambiar el contenido a imprimir de nuestro archivo .c original “HelloWorld.c” que inicialmente sólo imprimia: “HelloWorld”, y no nuestro cambio: “HelloWorld, Hello my love”, tal que así:*



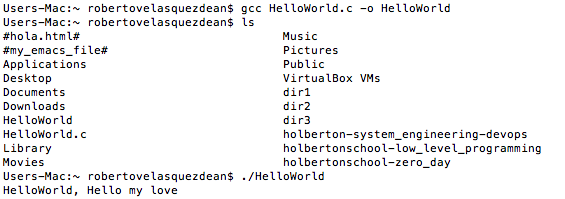
***Luego de guardar estas modificaciones, para poder ejecutarlas, debe de nuevo compilar el archivo .c original para crear un nuevo ejecutable (en caso que desee mantener el anterior, debe pasarle otro nombre; si no desea mantener el anterior ejecutable, puede borrarlo con un comando “rm” o sobreescribir sobre él al pasarle su nombre de archivo a este nuevo archivo ejecutable).*** *Observemos esto:*

*Voy a ejecutar mi archivo “HelloWorld” sin antes haber compilado sus modificaciones recientes, veremos que pasa:*



*Pasa que no me imprimió mi nueva actualización, la cuál es: “HelloWorld, Hello my love”. Ahora sí vamos a compilarlo para que se puedan ver los modificaciones realizadas en caso que deseemos ejecutar los cambios. Hay dos formas, como ya comentamos:*

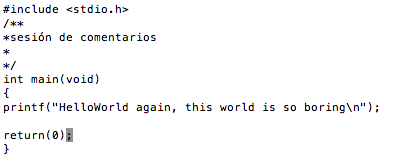
***Compilación de nuevas modificaciones borrando el actual archivo ejecutable:***



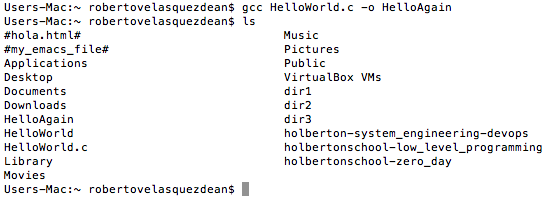
*Sí se da cuenta hemos “machucado” o sobre-escrito sobre el primer archivo ejecutable, lo cual lo elimina de forma automatica.*

***Compilación de nuevas modificaciones manteniendo el actual archivo ejecutable:***

*Vamos a editar, de nuevo, nuestro archivo .c original para insertar nuevos cambios, registralos en un nuevo ejecutable; pero, ojo, sin borrar el actual. Let’s go. Si por ejemplo deseamos cambiar el contenido a imprimir de nuestro archivo .c actual “HelloWorld.c” que ahora imprime: “HelloWorld, Hello my love”, a: “HelloWorld again, this world is so boring”, tal que así:*



*Pero queremos de todas maneras conservar ambos ejecutables resultantes; es decir, el que imprime: “HelloWorld, Hello my love” y el que imprimirá: “HelloWorld again, this world is so boring”. Debemos hacer un ejecutable, por medio de una nueva compilación, de este último con otro nombre (que no interfiera sobre el nombre del ejecutable actual), tal que así:*



*Se da cuenta que se pudieron conservar ambos archivos ejecutables resultantes a partir del mismo archivo .c original? Genial!*

***Aclaraciones importantes:*** *decir “./HelloWorld” significa, esencialmente, ir a la carpeta actual de la computadora (.), del cuál estoy buscando un archivo ejecutable llamado “HelloWorld” (/HelloWorld)*

*El signo dolar ($) que se muestra en cada nueva linea de comando (listo para ejecutar uno) es sólo un recordatorio visual constante que nos dice que “es aquí donde puedes escribir comandos”.*

*Aparte, si tenemos esto:* ***printf(“hello, world”); “printf”*** *es la función y* ***“hello, world”*** *es el argumento o, también llamado, parametro: entradas de la función.*

***Opciones del compilador GCC***

*Las opciones se ejecutan dentro del archivo origen tipo .c, no sobre el ejecutable resultante. Exactamente se ejecutan en la misma linea de código (y al mismo tiempo) en que se ejecuta el compilador de GNU (GCC) para justamente crear, ahora sí, un programa ejecutable resultante (o pueda que no se crea, eso depende también de que opción de GCC usemos) Vamos allá:*

***1. Opción “-Wall”:*** *Esta opción habilita todas las advertencias configuradas en GCC. Es decir, es la opción que nos señala todos los tipos de errores o advertencias que hay en nuestro archivo de origen .c al intentar ser compilado. Si hay algo mal en nuestro código de lenguaje C, esta opción se encargará de avisarnos.*

***gcc -Wall file.c (sinopsis),****Si no quiere que le dé nombre por defecto (a.out), puede darle uno con -o. Tal que así: gcc -Wall file.c -o name\_file*

***2. Opción “-E”:*** *Esta opción produce sólo la salida del* ***preprocesador.*** *La salida de la etapa de preprocesamiento se puede producir utilizando esta opción.*

***gcc -E file.c > file.i (sinopsis)***

*El comando gcc produce la salida en “Standard Out” para que pueda redirigir la salida en cualquier archivo (es decir, acá también podría usted mismo pasarle un nuevo nombre al archivo .c restante con la opción -o). Para nuestro ejemplo, el archivo* ***file.i*** *contendría por defecto la salida preprocesada.*

***3. Opción “-C”:*** *Para producir solo el código* ***compilado (sin ningún enlace).***

***gcc -C file.c > file.o (sinopsis)***

*El comando anterior produciría un archivo* ***file.o*** *que contendría código a nivel maquina o el código compilado.*

***4. Opción “-S”:***  *La salida de nivel de* ***ensamblaje*** *se puede producir con esta opción.*

***gcc -S file.c > file.s (sinopsis)***

*En este caso, el archivo* ***file.s*** *contendría la salida del ensamblado.*

***MÁS INFO, visite: [https://www.thegeekstuff.com/2012/10/gcc-compiler-options/](https://www.thegeekstuff.com/2012/10/gcc-compiler-options/ )***

*notas: La libreria fundamental de C es: “<stdio.h>” (Standard Input Output.h, significa: “darle capacidad a programas .C de obtener entradas y salidas del usuario”): por lo general todas las librerias que necesitamos se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Sin embargo, hay excepciones. Hay casos en los que deseamos “importar” una libreria que nuestro sistema no posee; tendríamos que descargar la libreria en cuestión y agregarla a la carpeta “libs” de nuestro entorno de trabajo, como por ejemplo: “VS code”. La función de las librerias es darle acceso a más funciones de las que podría obtener automaticamente del idioma que está utilizando.*

*Nuestro compilador será: gcc y el editor que usaremos para crear archivos tipo c será vi o emacs. GCC (GNU Compiler Collection) es un compilador integrado del proyecto GNU para C, C++, Objetive C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.* ***MÁS INFO, visite: https://www.thegeekstuff.com/2011/10/c-program-to-an-executable/***

*Como ya se dijo,* ***en el editor vi de la terminal podemos también crear archivos ejecutables tipo .c, los cuales pueden ser compilados en un compilador interno de GNU (sistema operativo) que se llama: GCC compiler, es nuestro medio para ejecutar C en Linux.***