***Programas de asistencia al código desarrollados por Harvard a través de CS50.***

***Help50*** *es un programa propio de cs50 instalado inicialmente en el entorno de trabajo de cs50 IDE que tiene la función de señalarnos posibles errores del tipo de "corrección" en nuestro código; es decir, nos ayuda a encontrar los errores propios al funcionamiento correcto de nuestro código, sólo le importa que el código funcione y sea ejecutable más allá de su eficiencia o de su estética (no evalúa diseño ni estilo). Se ejecuta a la hora de compilar con el comando "make" nuestro archivo originario.*

***Style50*** *tiene también los mismos orígenes en su creación que Help50, sin embargo, Style50 como programa tiene la función de señalarnos posibles errores del tipo de "estilo" o que afectan a la legibilidad visual: evalúa lo que se conoce como "las buenas prácticas", la estática del código, que esté bien escrito y "bonito" más allá de que simplemente funcione (no evalúa correción ni diseño). Se ejecuta sobre el archivo originario (código fuente, .c)*

***Check50****, al igual que Help50, es originado por cs50. Check50 tiene la función de señalarnos, en términos generales, si su archivo originario existe, si puede compilarse y ejecutarse, y si lleva a cabalidad bien unos requerimientos exigidos según el problema o proyecto planteado por Harvard en cuestión. Es nuestro checker. Evalúa corrección, diseño y estilo. Todo en su conjunto.*

*Hay otra herramienta,* ***debug50****, para ayudarnos a depurar programas. Esta es una herramienta escrita por personal que se basa en una herramienta estándar llamada* ***gdb.*** *Ambos depuradores son programas que ejecutarán nuestros propios programas paso a paso y nos permitirán ver variables y otra información mientras nuestro programa se está ejecutando. Los* ***Bugs*** *son errores o problemas en los programas que hacen que se comporten de manera diferente a lo previsto. Y la* ***depuración*** *es el proceso de encontrar y corregir esos errores. Los depuradores se asisten de algo llamado:* ***punto de interrupción,*** *línea de código especifica donde queremos que se detenga el depurador porque creemos que debiera detenerse ahí y analizar la línea de código en cuestión.*

***Para tener en cuenta:*** *los posibles errores del tipo "diseño", ya que no los mencionamos, evaluan qué tan eficiente es nuestro código y qué tan sofisticado es en su estructuración; es decir, ¿podría su código ser más "resumido" sin dejar de mostrar por pantalla la misma salida deseable o su código está repitiendo códigos innecesariamente?*

*Idealmente cada programa debe ser ejecutado en cada proyecto en cuestión antes de ser cargado; pues, todo proyecto pasará por un proceso de evaluación y auditoria. Dicho esto, ejecutarlos daría paso a una especie de retroalimentación para saber si nuestro trabajo está estrictamente bien hecho o no antes de ser subido.*

***Libreria y funciones Harvard:*** *La libreria de Harvard es:* ***cs50.h****, la cual se compone por las siguientes funciones correspondientes a obtener entradas de varios tipos de datos:*

***get\_char*** *(solicita por tecla un caracter al usuario), inicialización de variable:* ***char var = ...;***

***get\_float*** *(solicita por tecla un número con decimales al usuario), inicialización de variable:* ***float var = ...;***

***get\_double*** *(solicita por tecla un número con decimales de mayor peso en bits al usuario), “”:* ***double var = ...;***

***get\_int*** *(solicita por tecla un número entero al usuario), inicialización de variable:* ***int var = ...;***

***get\_long*** *(solicita por tecla un número entero de mayor peso en bits al usuario), “”:* ***long var = ...;***

***get\_string*** *(solicita por tecla una cadena de texto al usuario), inicialización de variable:* ***string var = ...;***

*Para* ***printf****, también, hay diferentes marcadores de posición para cada tipo:*

***%c para caracteres***

***%f para números con decimales (float/double)***

***%i para enteros***

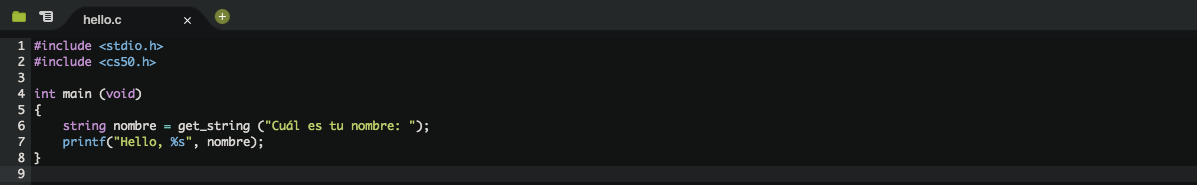
***%li para enteros mucho más grandes***

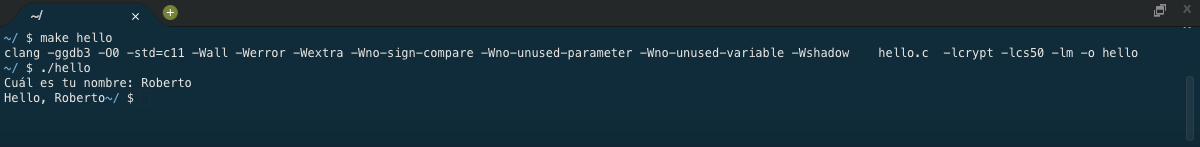
***%s para cadenas de texto***

la palabra “***var***” se refiere al nombre asignado para cada variable inicializada en cada tipo de dato.

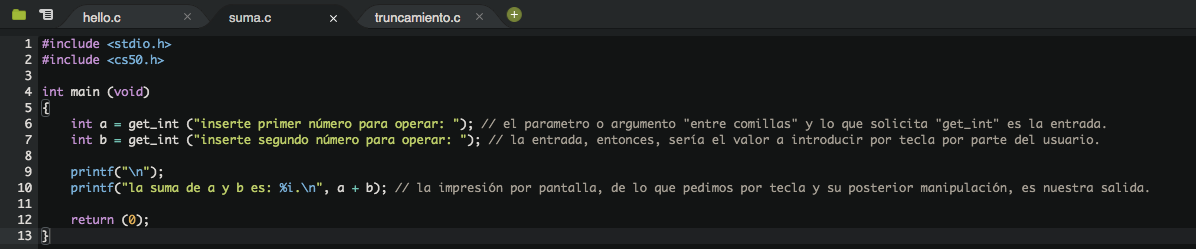
*No se olvide incluir la libreria de estas funciones, su código fuente debe comenzar así:* ***#include <cs50.h>***

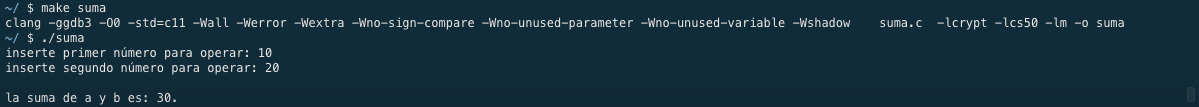
***Un ejemplo de cómo funciona la sintaxis de estas nuevas funciones agregadas a nuestro entorno de trabajo:***

código fuente (archivo originario de tipo “.c”): el que requiere ser compilado a código maquina.

código maquina (archivo ejecutable): proceso de compilación llevado a cabo y posterior ejecución del archivo.

*El anterior ejemplo fue realizado con una cadena de caracteres (string), ahora veremos otro ejemplo pero con la suma entre dos números enteros (int):*

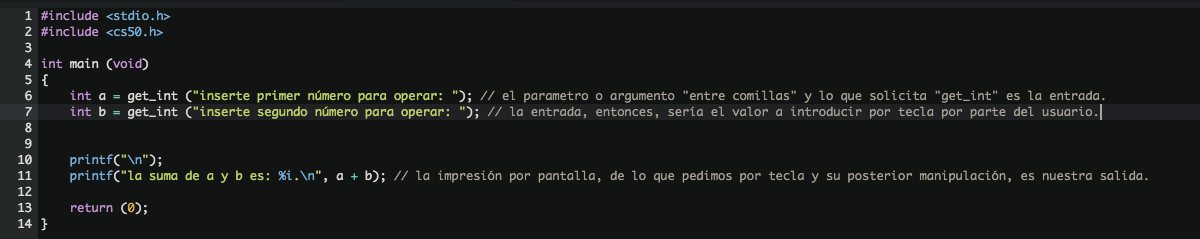
código fuente (archivo originario de tipo “.c”): el que requiere ser compilado a código maquina.

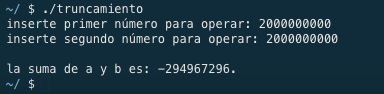
  
código maquina (archivo ejecutable): proceso de compilación llevado a cabo y posterior ejecución del archivo.

*Ahora revisaremos un nuevo y curioso caso de estudio que tiene que ver con los límites de bits con los que pueden trabajar los tipos de datos y sus funciones.* ***A veces sucede que, queremos manipular una variable de un tipo de dato en cuestión; pero, nos olvidamos de los límites, en términos de función y bits soportables, que posee el tipo de dato que está inicializando nuestras variables; y, entonces, nos extralimitamos con ellas.***

***¿Qué podemos hacer, por ejemplo, en el caso en que queramos manipular números enteros, declarados con “int”, que resultan ser demasiado grandes para la capacidad de memoria de dicho tipo de dato (int)? Lo ideal hubiera sido declararlos inicialmente como variable de tipo “long” para prevenir este tipo de situaciones; pero en caso que, en el camino nos hubieramos dado cuenta de que, las variables del tipo int declaradas efectivamente no iban a tener la capacidad de soportar una operativa de números más grandes que la propia capacidad de memoria que tienen ellas... Qué se haría?***

***Pues bien, para no vernos “truncados” en el camino, lo que se haría es una transición del tipo de dato, en el proceso, de las variables ya inicializadas: a ese proceso se le llama: “Casting”. Antes veamos este ejemplo:***

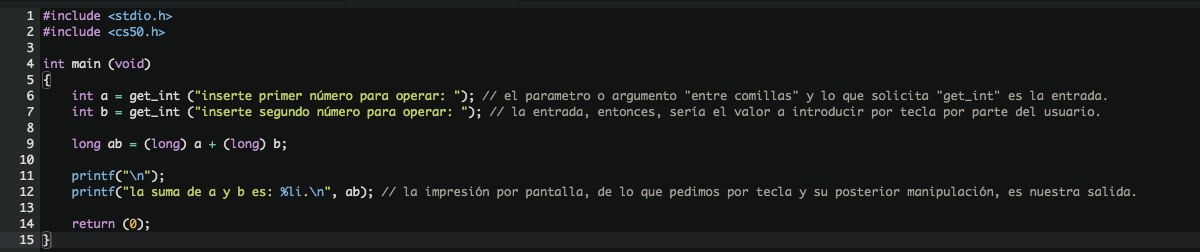




*Aquí hubo un “truncamiento”, si se fija la suma no está siendo correspondida:*

*Esto pasa porque el programa se “colgó”. Como las variables “a” y “b” fueron declaradas como números enteros de tipo “int” y dichos números sólo permiten una cantidad maxima de almacenamiento de valores cercana a los 2000000000 (de lado y lado, contando también sus negativos); entonces, al intentar sumar “a” y “b” el resultado esperado de la suma, como iba a ser superior a 2000000000 (exactamente 4000000000), se sobrepasarían los límites de la capacidad de memoria de la variable en cuestión, que es del tipo “int”... quedando así la terminal “loca”.*

*Para arreglar esto podríamos, en el proceso, convertir esas variables ya inicializadas como números enteros de tipo “int” a variables que sean de un tipo de entero con mayor capacidad de memoria; es decir, convertirlas al tipo de dato “long”. Esto se haría por medio del proceso ya mencionado: “Casting”. Tal que así:*



*si se da cuenta hemos declarado una nueva variable del tipo “long” con nombre de: “ab”; y,* ***el paso importante****, la hemos inicializado o asignado en ella ya el proceso de transición de las variables “a” y “b”,* ***que pasaron a ser variables del tipo “int” a variables del tipo “long”: “(long) a + (long) b;” (casting)****.* ***Con que se declare en “long” sólo una variable, dentro de la misma equivalencia, “(long) a” ó “(long) b”: todas las variables serán, instantaneamente, de tipo “long”.***

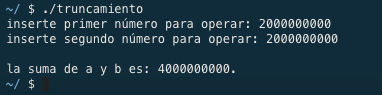
***Incluso, si inicialmente ya había sido declarada una de las variables en juego, “a” o “b”, como variable de tipo “long”, no hubiera sido necesario tampoco poner en práctica un proceso de “casting”.*** *Por ejemplo, en el siguiente escenario no hubiera sido necesario hacer “casting” para que el compilador interpretará que estamos, ahora, frente variables de tipo “long” y no “int” (incluida incluso la variable “a”):*

***int a;   
long b;***

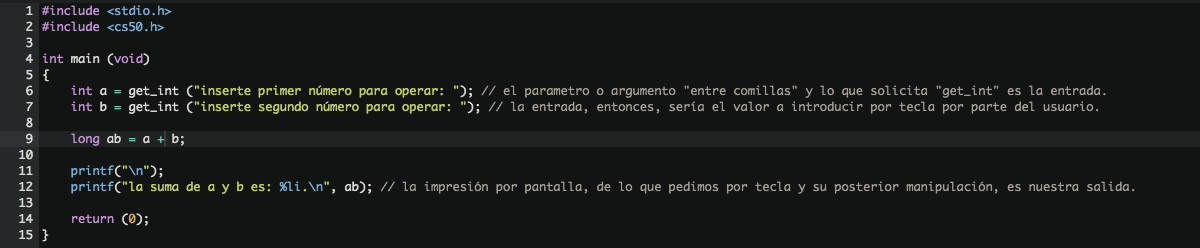
***long ab = a + b;***

***No es necesario hacer “casting” en este caso porque ya una de las variables de la equivalencia era, inicialmente, una variable de tipo “long”. Y con eso es necesario para que el compilador entienda que, entonces, la nueva variable “ab” es una variable de tipo “long”.***

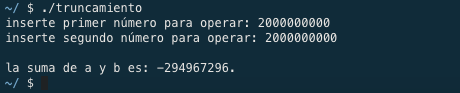
*Retomando nuestro caso puntual, el “casting” se hace con el motivo de que ahora el programa no se “crashee” por la falta de capacidad de almacenamiento en memoria en bits y por el límite de las funciones de un tipo de dato. Veamos que ahora sí nos corre el programa correctamente, pues, el tipo de dato “long” sí soporta números superiores a 2000000000 (de lado y lado, sean enteros negativos o enteros positivos):*



*No subestimemos* ***el paso más importante****, si no convertimos propiamente esas variables operativas (en nuestro caso, “a” y “b”) al tipo de dato ideal requerido, vamos a persistir en el error. Por ejemplo:*



*Acá sólo declaramos la nueva variable “ab” del tipo long y, aunque en la función printf la llamamos por medio del especificador de formato “%li”, no es suficiente; la terminal sigue interpretando que “a” y “b” son números enteros del tipo “int”. Y persiste el error:*

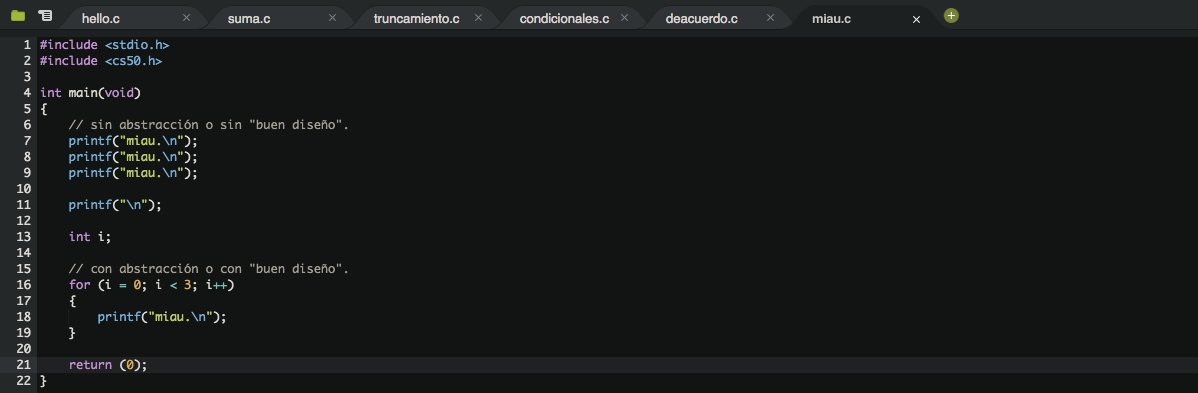


*Hay otro ejemplo de “truncamiento” que ya tiene que ver propiamente con las funciones especificas de los tipos de datos y no con su capacidad máxima de almacenamiento en bits. Puede revisarlo en la documentación original de Harvard notes, week 1 (C).*

***Qué es abstracción?***

*Parte de un principio de resolución de problemas que radica en “simplificar detalles que de otro modo serían más complicados”. Es decir, se conoce por “abstracción” hacer una simplificación de detalles que originalmente resultaban ser más extensos. Lo anterior podría relacionarse con las buenas prácticas por las que se compone la variante del “diseño” en la programación.*

*Podemos reflejar aquí un buen ejemplo de “abstracción” que, fuera de tecnicismos sería el equivalente a: tener un buen diseño. Veamos:*



*En contexto... puedo imprimir la palabra “miau” las mismas tres veces en una misma linea de código de una función y sin necesidad de ocupar, paralelamente, tres lineas de código para eso: de esto se trata la abstracción o el arte del “buen diseño”.*

***Qué es encriptar?***

***“encriptar”*** *significa ocultar algo de manera reversible (que pueda ser “des-ocultado”). El* ***texto no cifrado*** *generalmente se denomina* ***texto sin formato (texto plano)****. El texto encriptado generalmente se denomina* ***texto cifrado****. Y el secreto utilizado se llama* ***clave*** *.*

***Unencrypted*** *text is generally called* ***plaintext.*** *Encrypted text is generally called* ***ciphertext.*** *And the secret used is called a* ***key.***