

Universidad autónoma de baja California

Ingeniería en computación

Microcontroladores

Reporte practica 6: UART

Erik garcia Chávez 01275973

Jesus Adan Grcia

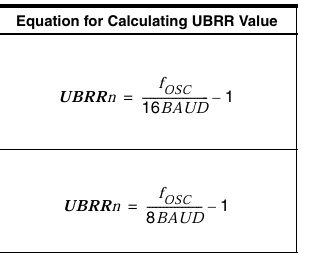
27 de marzo del 2025

**Teoría:**

***Manejo de UART:***

el atmega 2560 cuenta con 4 UART’s , el UART se compone de 3 “partes” tenemos el *clock generator, transmitter, receiver.* Los registros son comprartidos.

El clock del UART se maneja bajo baudios, para poder saber la velocidad de este reloj es necesario una formula, tenemos 2 variantes uno que opera normal y otro a velocidad doble.



Esta velocidad es con a que se estará comunicando con la terminal o incluso son otro UART’s,

Esta velocidad se establece bajo el resigtro UBRR el cual es de 12 bits que cuenta con parte baja y parte alta.

El UART tiene 4 registros, el UBBR, tiene *UCSRnA, UCSRnB y UCSRnC,*

En el registro *UCSRnA* los bits que nos improta en esta practica seria tan solo UDREn y U2X

*UDREn* :  *este bit indica que UDRn esta vacio por lo que est listo para recibir nuevos datos*

*U2X:* indica la velocidad, tenemos veldidad doble, con velocidad simple en la misma trama se mandan 16 muestras y con la velocidad doble se envían 8 muestras para la misma trama,

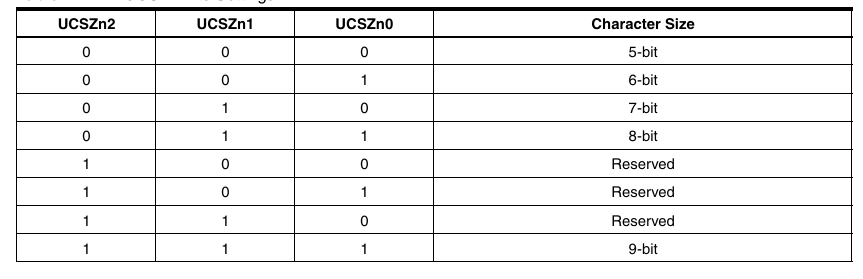
*Registro UCSRnB,* de igual forma los registros que por ahora nos interesan serán:

*RXEn, TXEn :* debemos habilitar estos bits para que el UART pueda funcionar como un puerto serial entonces ahora si puede recibir y transmitir, sus pines asociados pierden su propiedad digital y se vuelven un puerto serial

*UCSZn2: en este registro, si se activa quiere decir que la trama de la infromacion será de 9 bits.*

Registro *UCSRnC* bits que nos interesa, este registro es el que tiene mas bits para configurar nustro UART.

*UCSZn0, UCSZn1* : estos bits no dinican de que tamanio será la trama de la infromacion, con esto 2 bits se puede configurar que se pueden enviar hastas 8 bits de información, si se quiere mandar 9 bits es necsario el bit extra en el registro *UCSRnB.*



USBSn : indica el stop bit, este le dice al UART que hasta aquí finaliza la trama de infromacion, el cual pueden ser 1 bit o 2 bits

*UPMn1, UPMn0:* habilita el bit de paridad, el bit de paridad funciona para verificar posibles errores de trasmicion en la trama.

*UMSEL1, UMSEL0 :* este registro selecciona que modo de operación va a funcionar el UART, puede ser síncrono, asíncrono o master SPI, a nosotros para esta practica manejaremos el modo asíncrono.

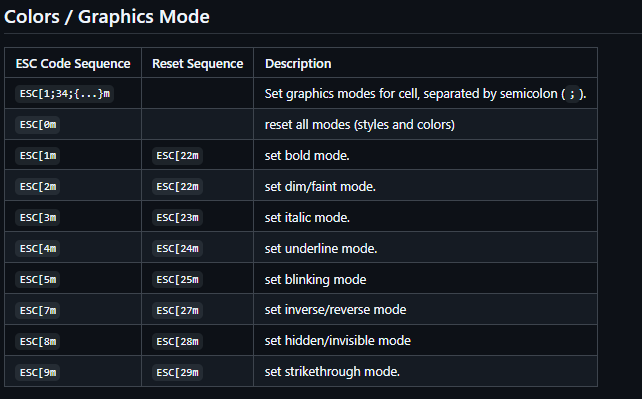
**Secueincas de escape ASNI**

las secuencias ASNI funcionan para especificar acciones como retornos de carro, movimiento de tabulación en terminales, también empleados para proporcionar representación literales de caracteres no imprimibles y de caracteres que normalmente tiene significados especiables.

estas son utilizandos principalmente en entorno de lina de comandos CLI para crear interfaces mas interactivas o visualemtne atractivas dado que se puede cambiar colores o posiciones dentro de la terminal

la estructura básica, esta secuencia comienza con el carácter **ESC (27 ascii),** seguido de corchetes **[** parámetros que estos pueden tener son serpados por **;** y una letra indica el comando, el carácter ESC puede ser represado como **\x1B[<parametors;P><comando>**

aquí tenemos unos ejemplos de estos:

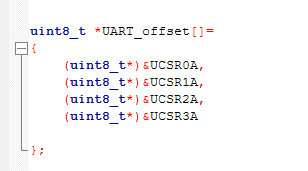


**EXPLICACION CODIGO:**

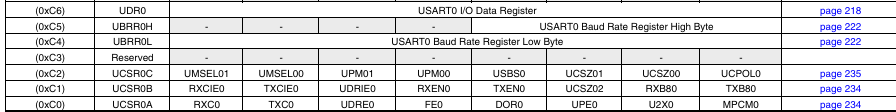


Como primera instancias tenemos nuestro archivo cabecera en donde se están declarando los prototipos y varibales globales que esteremos usando en el código, esta entre una directiva del preprocesador de C, esto se hace porque se esta usando la cabecera en varios archivos para no evitar errores se declara una vez las varibales y protoripos para que no ocurran errores durante la ejecución del código.

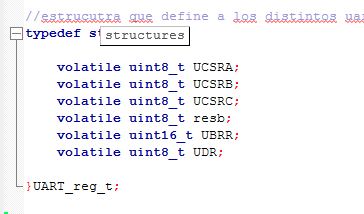
Para poder llamar a cualquier de los 4 UART’s que tiene el atmega, se usa un arreglo por medio de apuntadores, estamos llamando a la dirección de memoria de x UART, dato que los registros de los UART entre si tienen la misma distribución de los registros pero entre los espacios del UART 2 al 3, hay un gran espacio en memoria que se ocupa para algo mas, para ser mucho mas exacto y que el cosigo sea muy reducido, usamos apuntadores, traemos el primer registro del UART



Con este registro dependiendo de lo que el usuario solicite iniciamos en el UART solicitado, UCSRnA es el registro inical de cada UART. Podemos ver como esta distribuitos los registros en el atmega.



Para poder usar estos registros hacemos uso de una estructura, la estructura tiene la misma disponibilidad en memoria que como en el atemga.



Con esto vamos a poder cualquier UART, esto va a depender del usuario.

**UART\_Ini:**

la función que incializa el UART debe ser muy genérica, lo que recibe como parámetros es que UART el usuario quiere usar, se le pasa el un valor al baudaje que se quiere llegar al final ese valor no es el que se le asigna a UBRR si no que es necesario pasar por una ecuación que el valor de la velocidad con las que el UART se va a comunicar.

Recibe el tamanio del frame que se va a transmitir, asi como los bits de paridad que se va a querer y los stop bits, como estos son recibidos como 0, 1 o 2 en el caso del bit de paridad y 1 y 2 en el caso del stop bit hago usado de switch-case para los bits de paridad porque para 2 bits de paridad es 3 (0011) por lo que ocupamos cambiar lo nos manda a su reprsentacion adecuada para los bits en el registro lo mismo con los stop bits.

****

**SEND:**

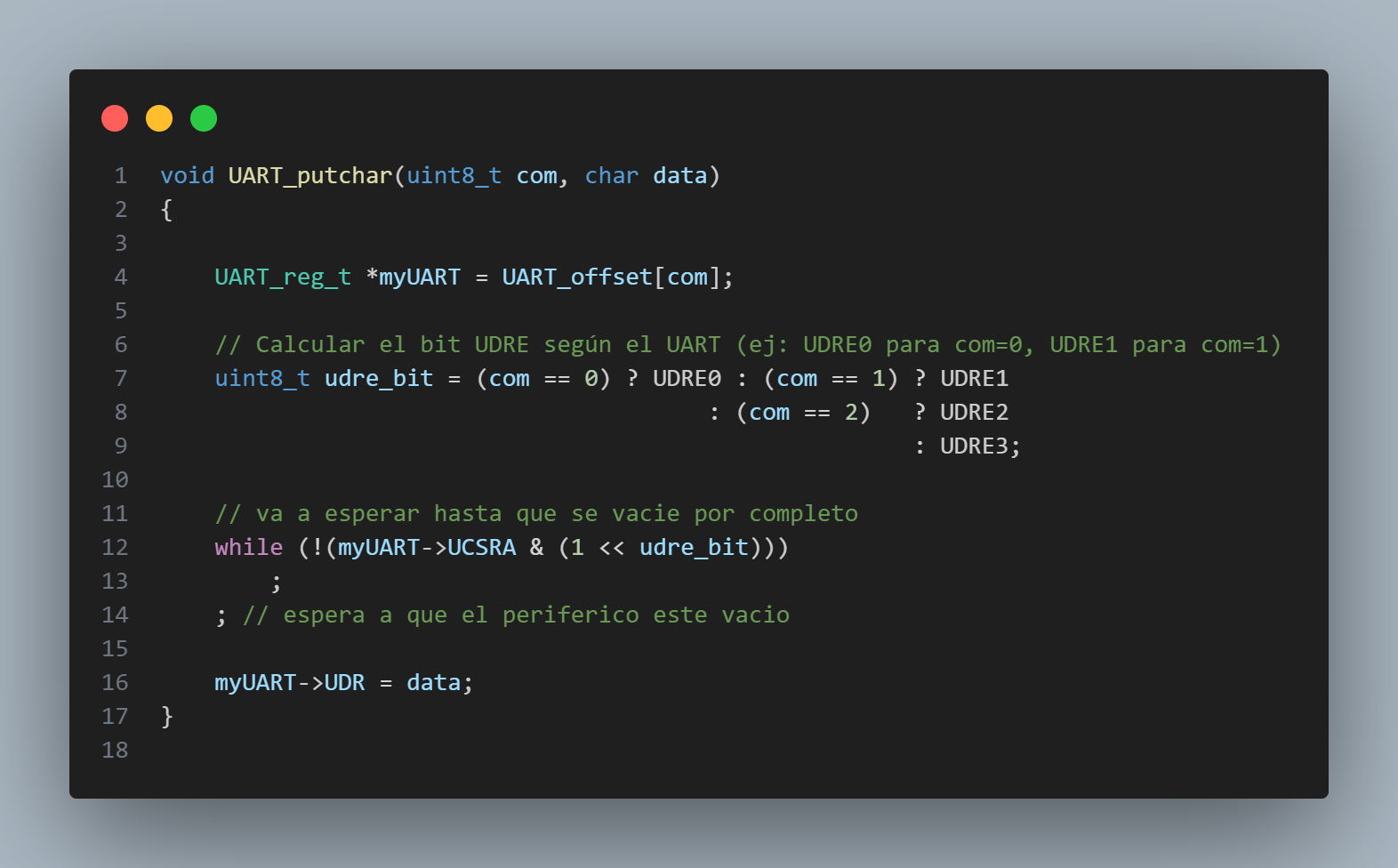
*UART\_PUTCHAR*

La función del putchar es la de enviar un único carácter (byte) a través del modulo UART. Es una función bloqueante lo que quiere decir es que espera hasta que el modulo UART este listo para transmitir antes de enviar el dato.

Como tuve problemas usando UDRE de manera genreica, medinadiante unas condiciones traemos el que le corresponde a cada UART. El UDRE es el bit de UCSRA que indica si el buffer de tranmiscion esta vacio.

Aplicamos una mascara a UDRE porque solo nos interesa ese bit, el registro UDRE esta en 1 cunado es buffer esta vacio, por lo que al aplicar una negación estamos diciendo que si esta lleno cunado esta en 0 este va a esperar hasta que se vacie, y cunado este esta vacio (1) pero con la negación 0 este empieza a trasmitir el contenido en el registro UDR

Lo que esta recibiendo es el puerto por el cual se va enviar el frame asi como el dato que se quiere enviar



**UART\_ puts:**

La función lo único que esta haciendo es mandar a llamar a la función *UART\_PUTCHAR* este lo manda a llamar hasta que en el string encuentre el carácter vacio lo que indica que el carácter llego hasta su máxima capacidad y es el final, como es un apuntador realmente no sabemos que de tamaño es por lo que agrego el carácter nulo ‘\0’ al final de la cadena para indicar ese final.



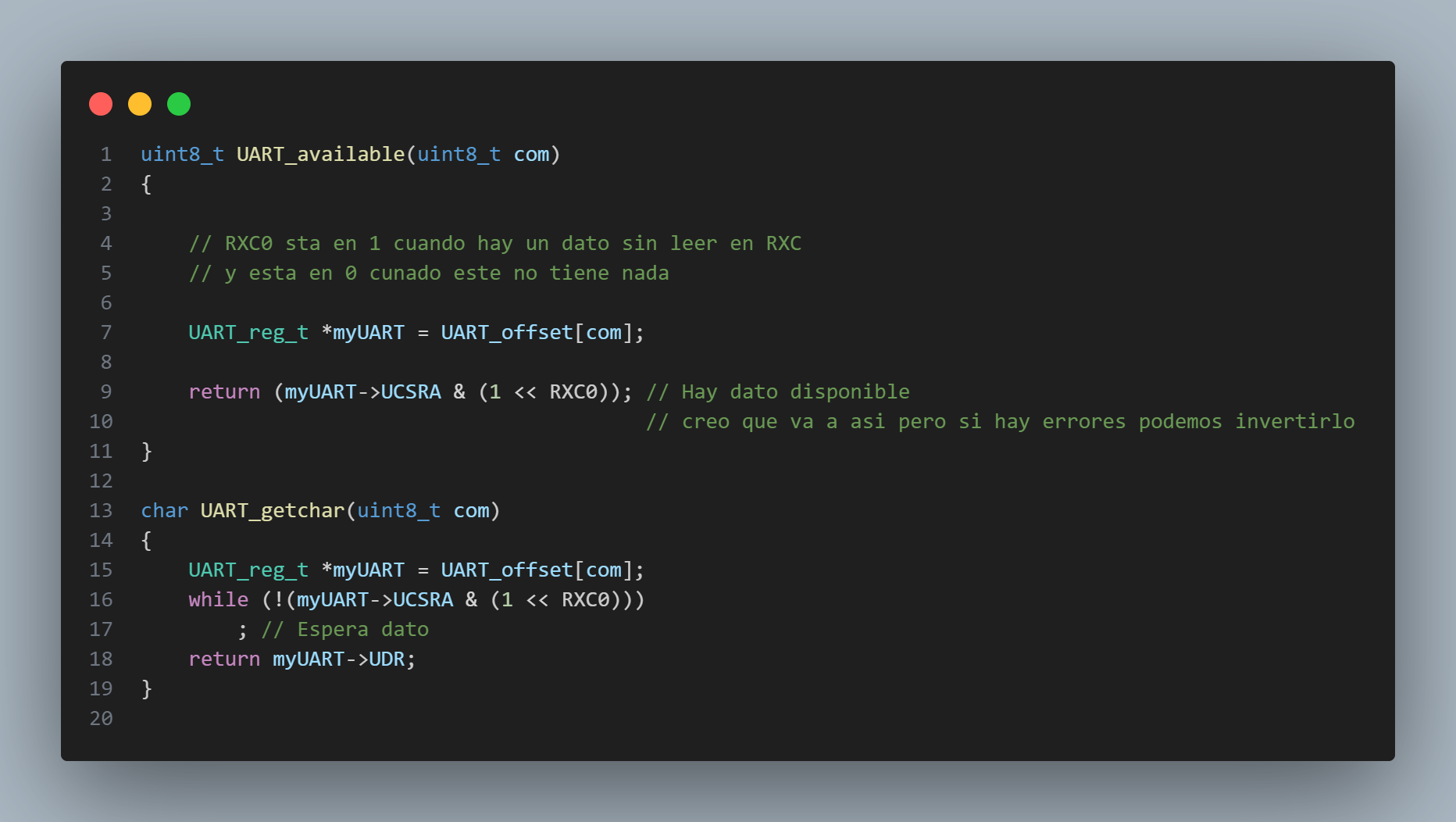
**RECIVED**

*UART\_getchar():*

Esta función se encarga de recibir un carácter (byte) desde un modulo UART especifico por esa razón solo recibe como párametro el puerto del cual va a leer

Mediante la estructura y el arreglo seleccionamos el UART que vamos a usar.

Esta función espera hasta que reciba un dato, haciendo una mascara al bit RXCn (USART RECEIVE COMPLET) el cual esta en alto cunado hay un dato sin leer dentro del buffer y esta en 0 cunado esta vacio. Por lo que al negar esta condición le estamos indicando que espere meintras RXC no tenga datos disponibles, cunado RX reciba un dado la condición será 0 por lo que no entra al bucle y pasa a leer UDR retornar el valor.

**

*UART\_gets():*

Esta función tiene la función de recopilar todos los caracteres hasta un tamanio especifico asi como fungir como arbitro dependiendo de los diretentes sucesos que pueden ingresar por la terminal. Tenemos vairas partes

En primero tenemos un super loop, esto porque no sabemos hasta cunato es el buffer que tiene en este momento, tenemos varias varibales que funcionan de la sigueite manera, **C** funciona para capturar el dato que se recibió pro *getchar,* **i** funciona como el índice que indica en que posición y que tamanio tiene en este momento el apuntador del string captruado por el UART, **dot\_flat** es una varibale especial que uso en el caso que se introduzcan puntos.

Lo primero que se hace es captruar el carácter que se recibió *UART\_getchar,* en el caso que el usuario decida borrar un carácter es la primrea condición que tenemos, esto porque, porque si el usuario borra algo es lo primero que debe de pasar antes de mostrar, el carácter **‘\b’** me indica que se esta presionando la tecla *backspace* pero tenemos una condición en el caso que **i>0** no queremos que borre lo que esta en la terminal en este caso seria una instrucción, porque eso no forma parte de lo que el usuario ha introducido por lo que lo que hacemos es pasar de largo y continuar con el programa pero en el caso de que el usuario haya ingresado algún elemento, va a retroceder el índice dado que se esta liberando ese espacio **str[--i] = ‘\0’** perimero se retrocede y después se remplaza el ultimo carácter con el carácter nulo

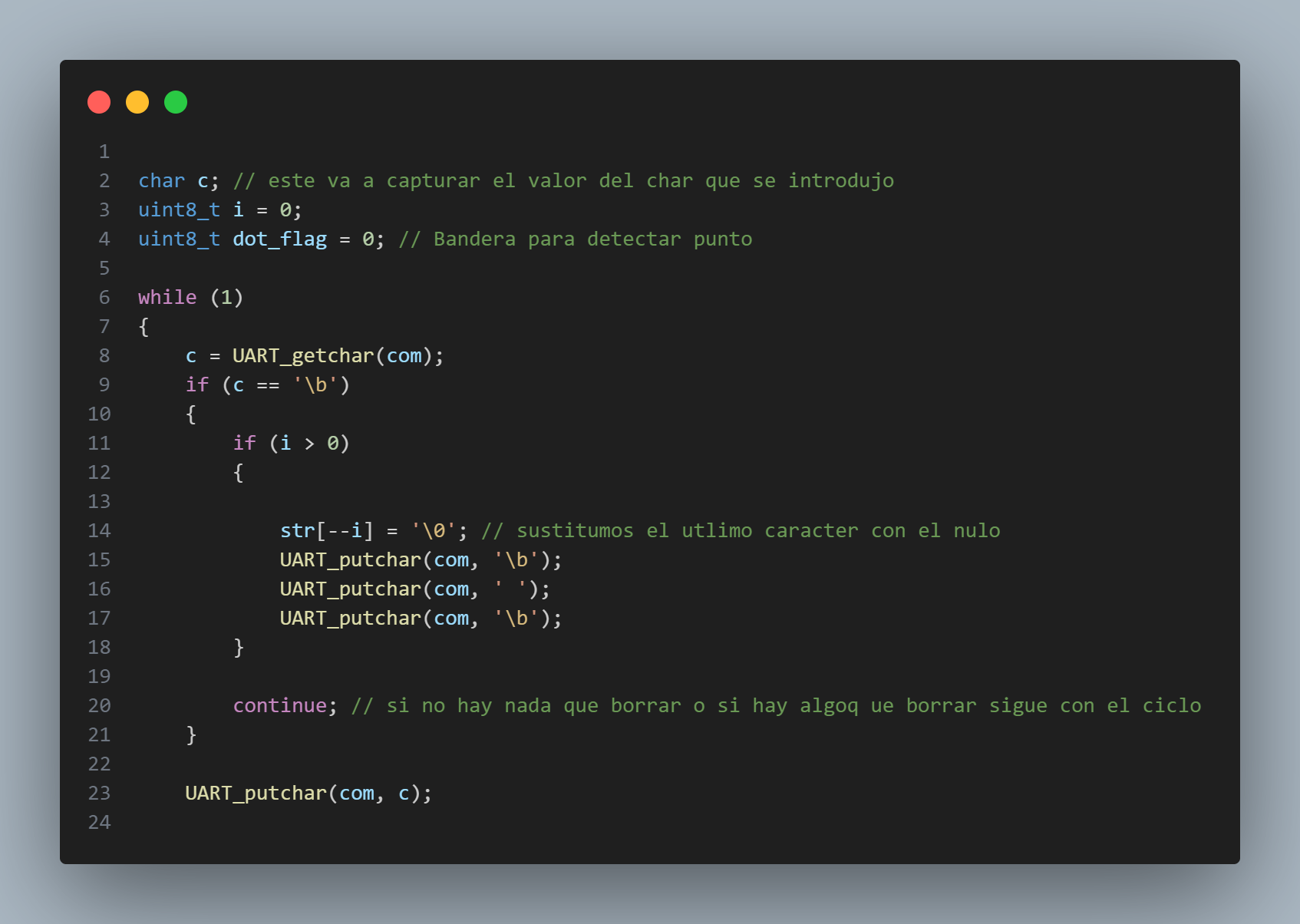
Con:

UART\_putchar(com, '\b'); Mueve el cursor a la izquierda.

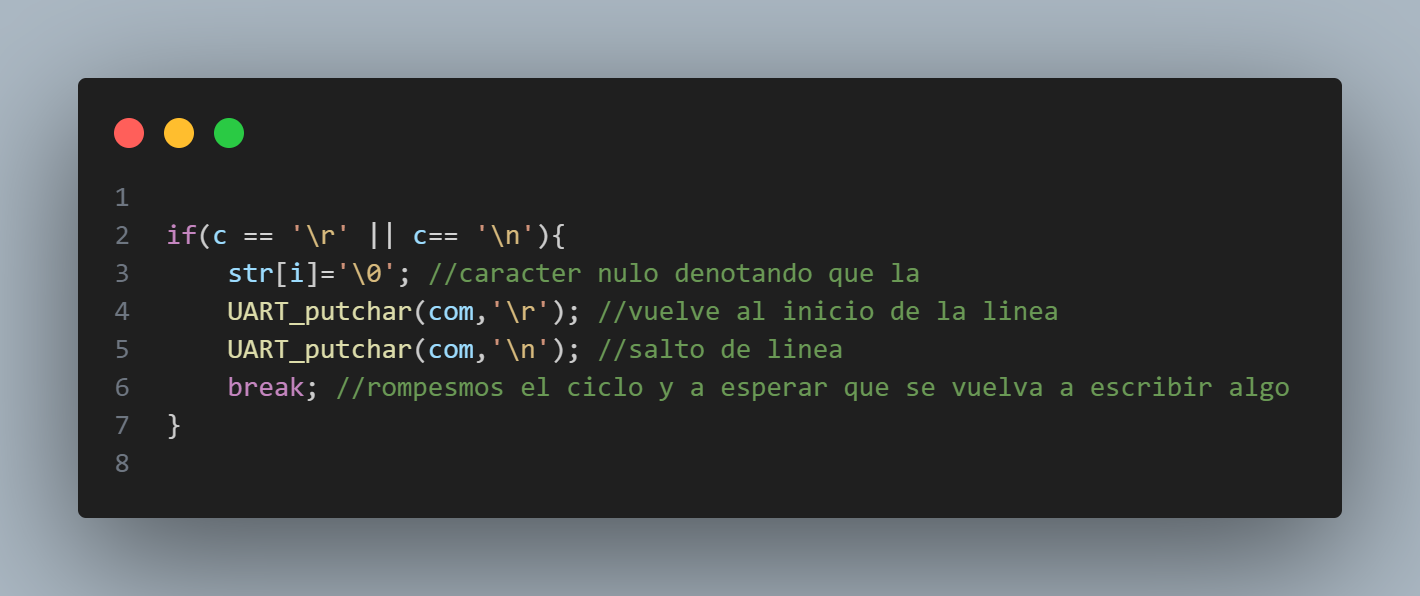
UART\_putchar(com, ' '); Sobrescribe el carácter con un espacio.

UART\_putchar(com, '\b'); Mueve el cursor a la izquierda nuevamente.

Saliendo de la condición mostramos por pantalla lo que tiene **C.**

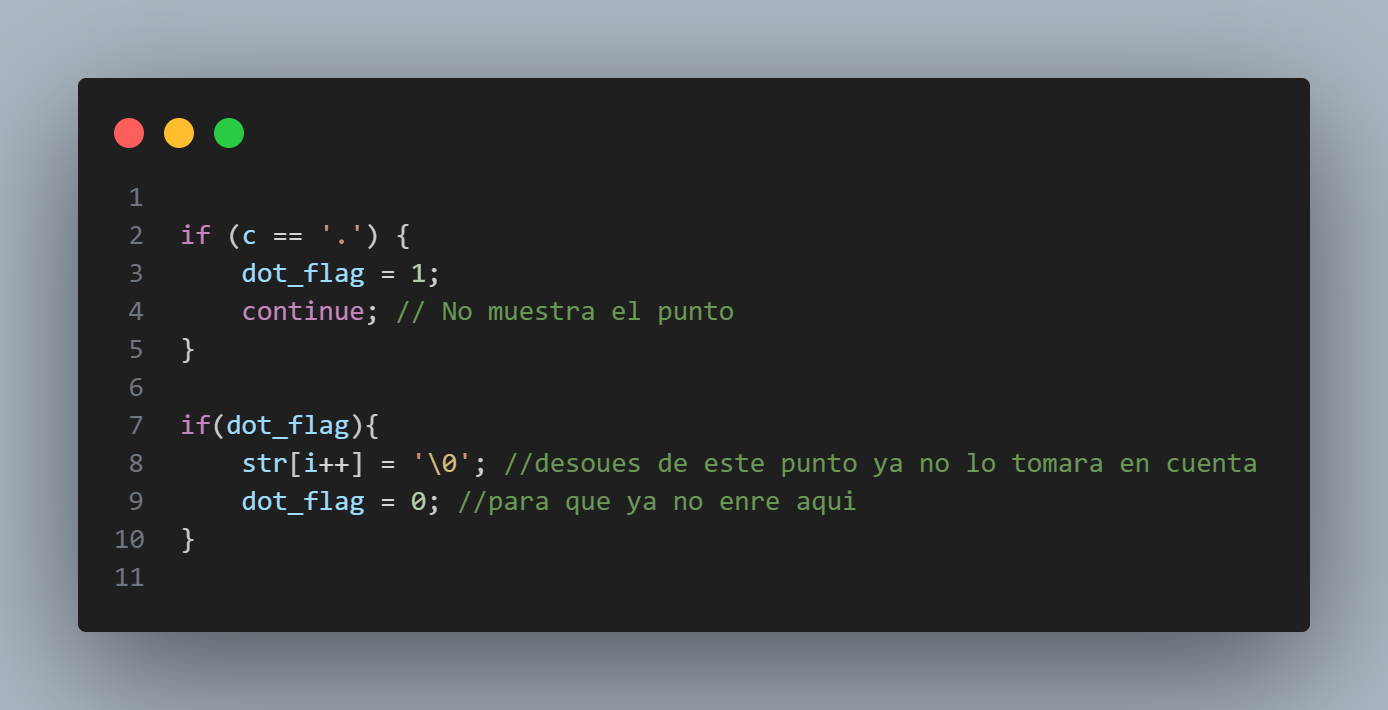


Como saber que la cadena ya termino, una parte claro el carácter vacio ‘\0’ pero como saber que la función gets ya dejarla, porque como se ve es un ciclo infinito, por lo que la forma de saber que ya no queremos ingresar nada mas es con el carácter ‘\n’ que seria **enter** en nustro teclado. Por lo que estamos constantemente buscando lo que esta introducciondo por UART pero si es el carácter de salto de línea quiere decir que el usuario ya termino de ingresar los datos, por lo que finalizamos la cadena con el carácter vacio en la posición de **i** asi que hacer un retorno de carro para ir al inicio de la terminal y brincamos a la línea siguiente.



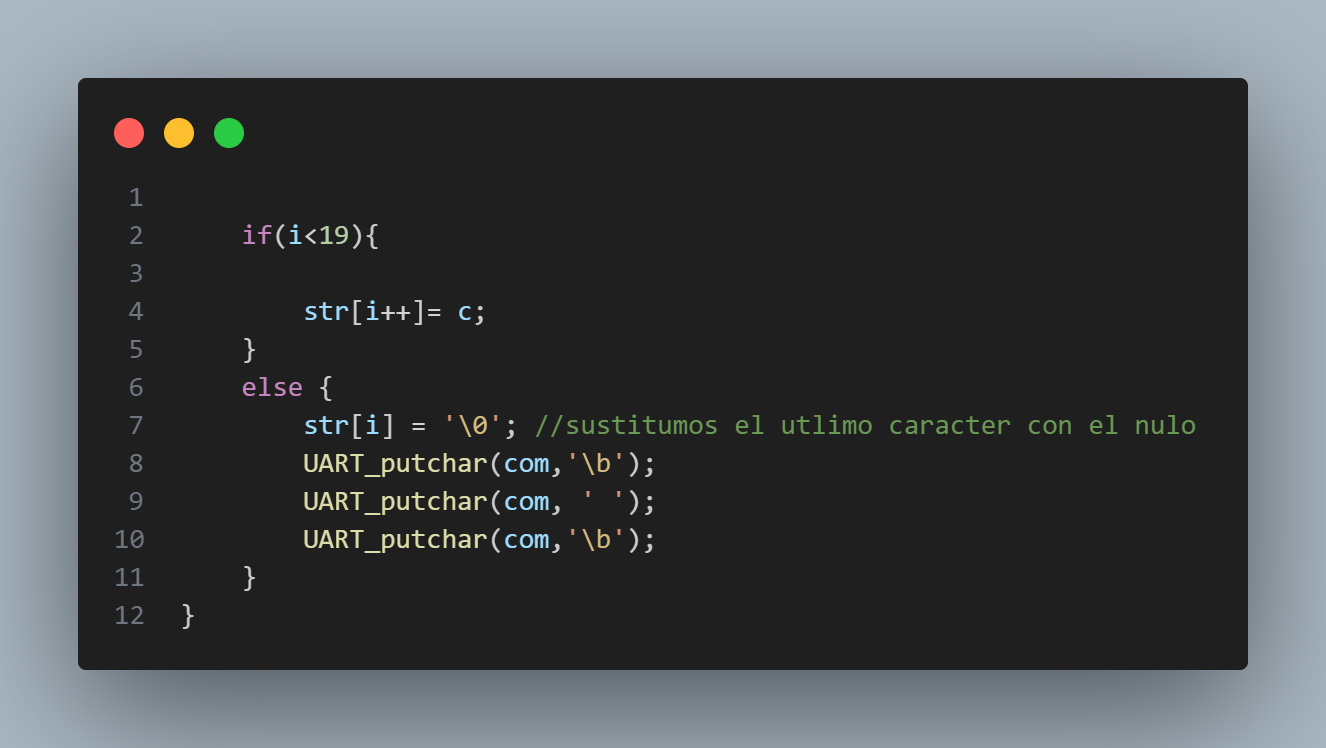
**Caso especial de ( . )**

Evitamos procesar numero con punto decimal para evitar posibles errores, por lo que la idea es que si mira un punto entre la infromacion ingresada tengo una bandera que indica si se ingreso un punto, en la condición donde entra cunado la bandera esta activada ponemos el carácter nulo para terminar donde esta el punto, esto porque sabemos que el puts imprime hasta que enceuntre el carácter nulo, esto se hace para que lo que este depues del punto lo ignore, se imorimre por UART pero esta haceindo ignorado para ser procesado con **itoa** y **atoi,** después la borramos para que ya no entre no necesitamos mas porque con el único donde esta el punto es ams que suficiente.

****

**Limite de introducir.**

Ponemos un limite de los datos que se muestran por UART, verificamos que el índice sea menor a la cantidad limite, esto porque el ultimo índice estaría reservado para el carácter nulo ‘\n’, para que no siga mostrando lo que se hace es algo parecido al borrado, constantemente estamos remplazando ese nuevo carácter con un espacio vacio y retorcedemos cursor constantemente

****

**

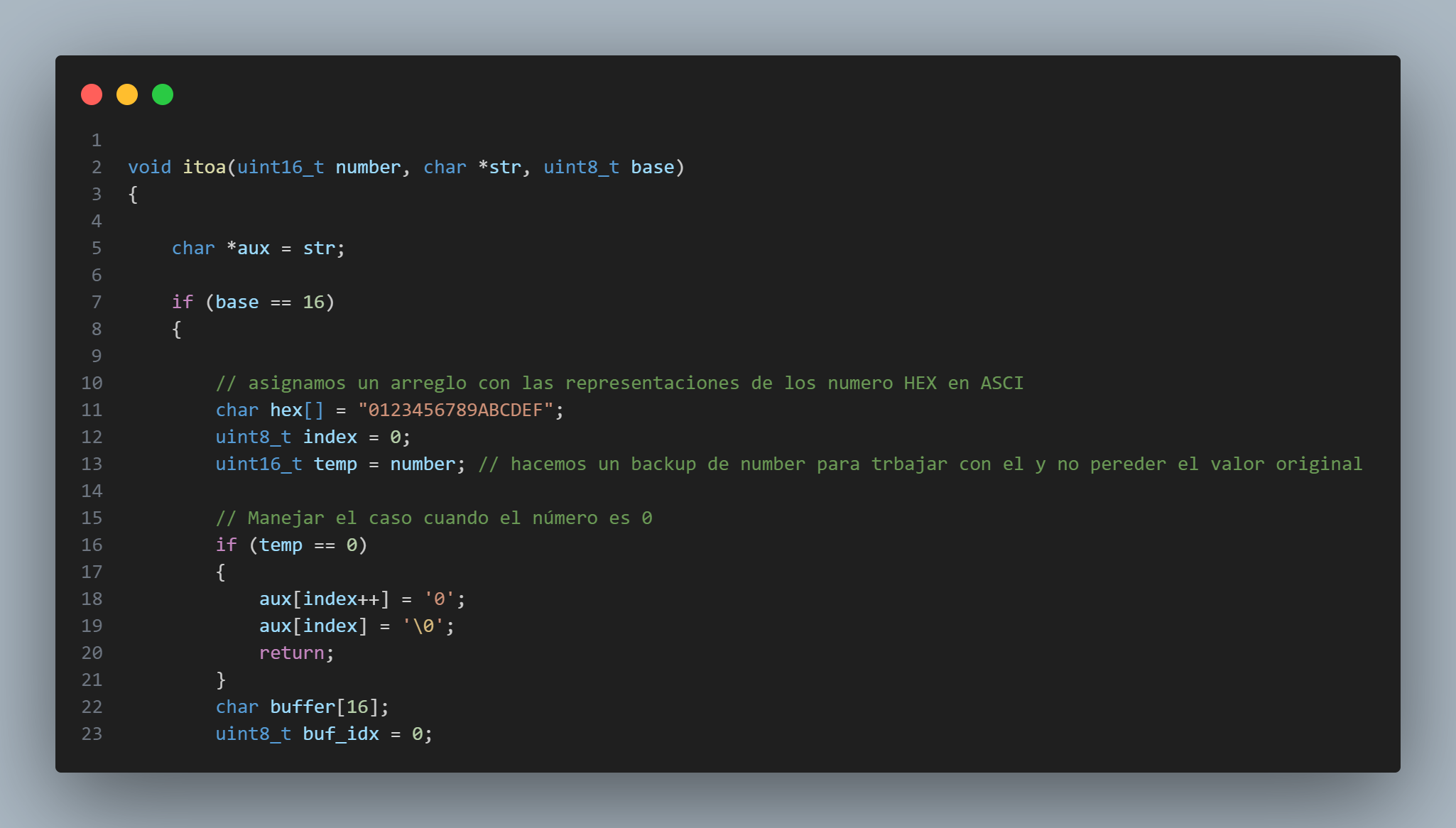
**Itoa:**

Itoa convierte un valor numerico a su representación HEX pero en ASCII para poder ser mostrado por UART. La función convierte tanto binario como HEX por lo que estará recibiendo el numero que anteriormente se convirtió en atoi, el apuntador de la cadena asi como la base a la cual se quiere convertir, se utiliza una varibale auxiliar para no perder el puntero inical de la cadena.

Hacemos una matriz la cual tiene las letras que reprsentna los números en HEX, asi como una varibale que tiene la cantidad igual que **number** esto para que el valor original no se pierda.

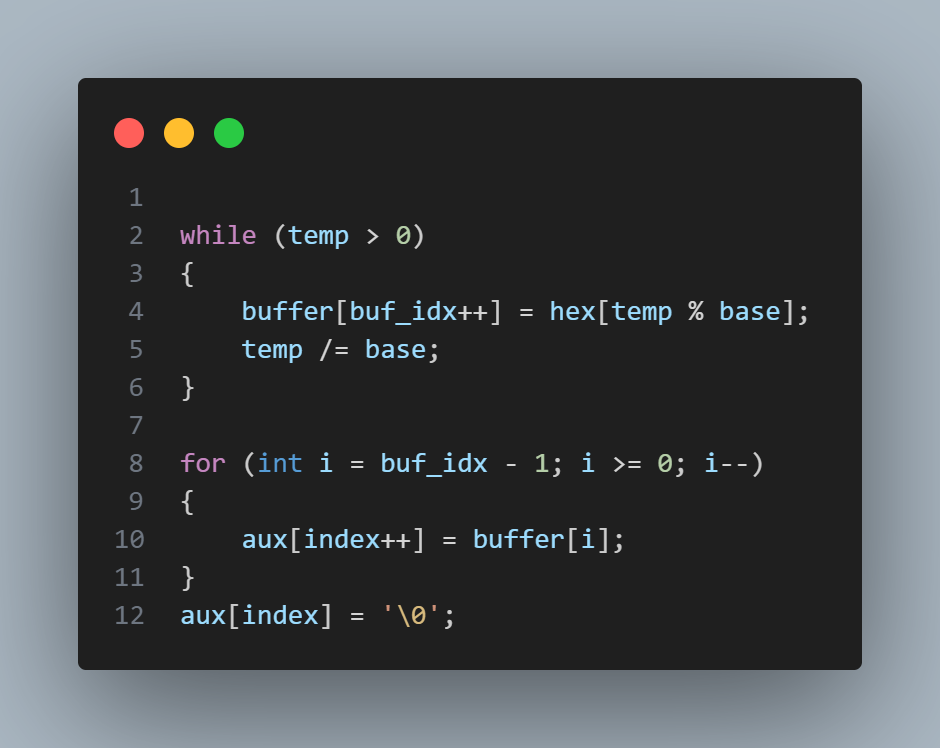
Primero verificamos que el numero no sea cero, si es asi no tiene caso el que se haga un proceso que solo seria tiempo perdido.

Asignamos un arreglo **buffer** que es donde se estará guardando la represnetaciones HEX en ASCII del numero que estamos conviertiendo asi como un contador que funcionaraia como que tamanio es el digito.

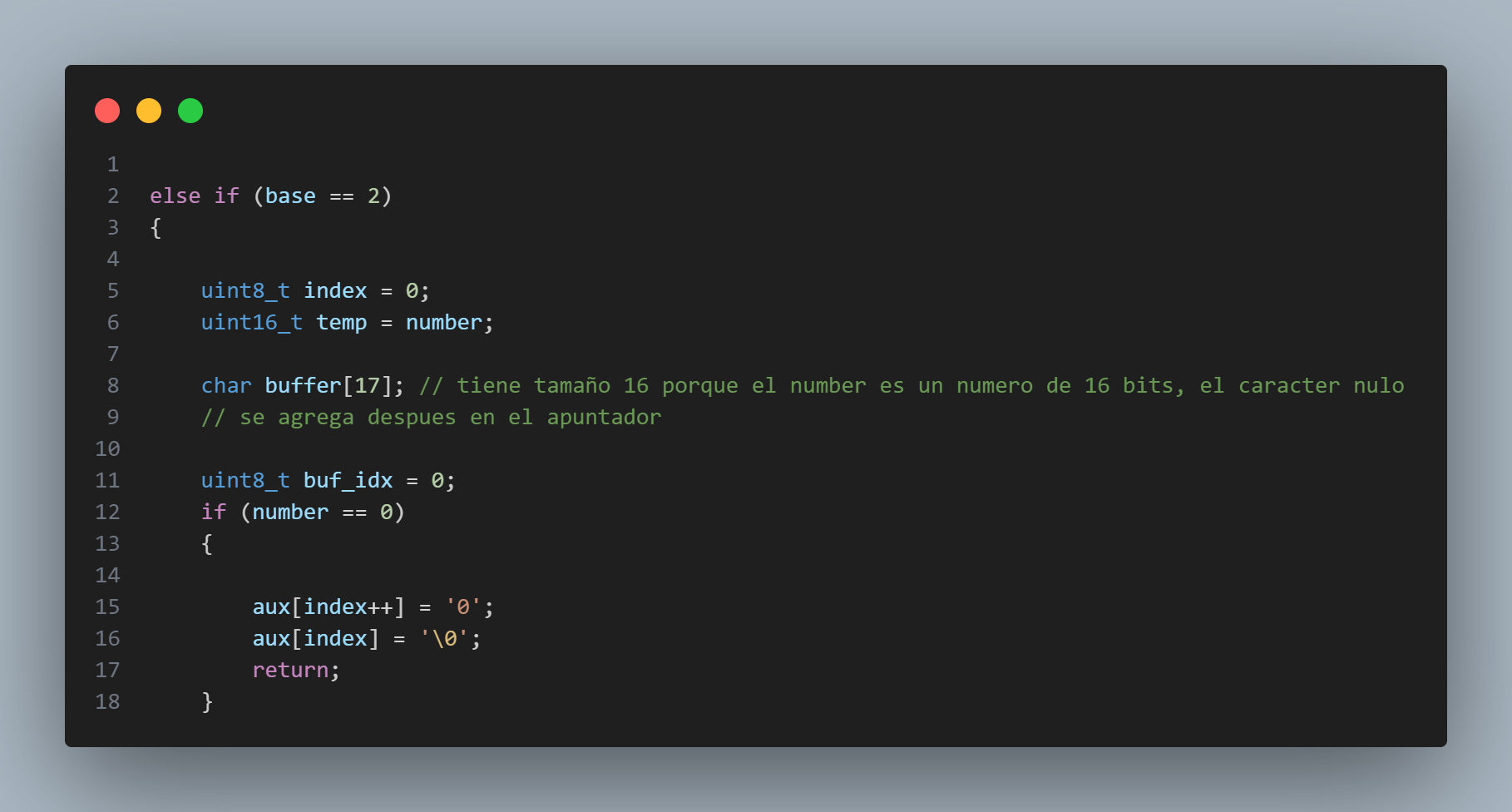


La operación correcta para calcular la representación HEX de un numero decimal es mediante la división modular, es decir el resto de la división, el resto de esa división es la representación del numero decimal, se divide dependiendo la base, para HEX entre 16, el numero resultante es exactamente el índice en nuestro arreglo **hex[]** es decir que el resultado de la división modurar es 11 el inidce 11 de nuestro arreglo es el **B** por lo que se selecciona ese elmento, cada elemento se va agregando al segundo arreglo el llamado **buffer** el cual funciona para alamcenar cada carácter, vamos reduciendo el numero hacindo una división entera sobre la misma base

Como sacamos los valores ahora?, con este método estamos haciendo como una pila, esto porque porque el primer elemento que se calcula es el ultimo elemento en la representación del numero HEX, es decir si se calcula **[B,0,2]** que seria la repsetnacion HEX de 523 pero **B02** realmente no es **523** el numero esta al revez, por lo que es como una pila primero se saca el ultimo elemento en entrar a esa pila, para eso hacemos uso de un ciclo for dado que conocemos la longitud del numero que del tamanio de **buf\_idx** como la cadena es un apuntador empezamos a remplazar desde el inicio y al final volvemos a agregar el carácter nulo denotando el final de dicha cadena.

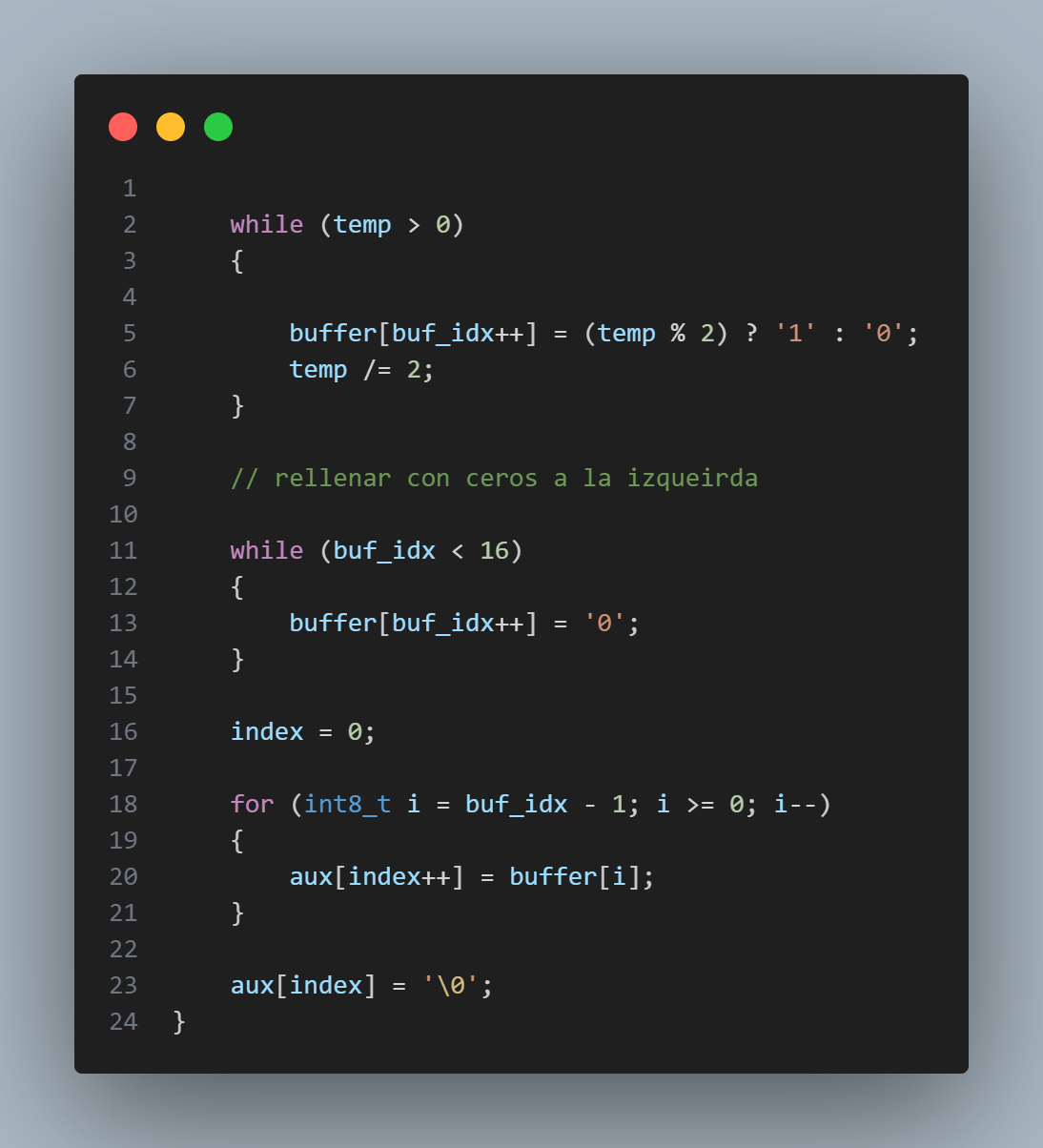


Para cuando es binario es algo parecido, al inicio tenemos nuestra condición en caos que sea cero, asi como nuestras variables locales que son copias de los argumentos, asi común arreglo que funciona la **buffer** en donde se guarda el numero en su rerpesentacion binaria.



La manera de calcular si es 1 o si es 0, es muy parecido al de HEX usando la división modular, pero esta es diferente, si el resto es 0 es obvio que el bit es ‘0’ pero si es diferente, como 5 entonces el bit seria ‘1’, por lo que hacemos lo mismos de agregar cada bit al buffer, e igual que con HEX el ultimo bit en entrar es el MSB y el primero en entrar es el LSB.

Agregamos ceros una posición después de en donde se quedo el índice el buffer, porque no se sabe que puede haber después muy posiblemente haya basura y no queremos eso.





**Atoi:**

Esta función lo que hace es recoge los valores numéricos en ASCII y los convierte en su valor numerico. Hacemos uso de una varibale que funge como acumulador

Lo que se hace es recorrer el string hasta llegar al final **‘\0’** mientras que este entre el valor numerico decimal en carácter

La opreacion que se hace es primero calcular en su vesion numercia que numero es el que se esta tratando, por ejemplo si tenemos ‘1’ en su verion ascii serai 49, a todos los números si le restamos 48 -> ‘0’ el resultado seria su valor pero en versión numérica, 1, 2, 3,., etc. Pero como saber si el digito que sacamos de esa resta corresponde a una decena, una centena, etc. Lo que se hace es que al resultado pasadao lo multiplicamos por 10, esto para calcular su verdadero valor, y todo se va sumando por ejemplo si tenemos ‘123’

Calcula:

0 \* 10 + 1 = 1

1 \* 10 + 2 = 12

12 \*10 + 3 = 123

