|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Karina García Morales |
| *Asignatura:* | Fundamentos de Programación |
| *Grupo:* | 20 |
| *No de Práctica(s):* |  |
| *Integrante(s):* | Miguel Oswaldo Noguez González  7 |
| *No. de Equipo de cómputo empleado:* | NA |
| *No. de Lista o Brigada:* | 36 |
| *Semestre:* | 2021-1 |
| *Fecha de entrega:* | 25 de noviembre del 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**FUNDAMENTOS DEL LENGUAJE C**

**Objetivos:**

Elaborar programas en lenguaje C utilizando las instrucciones de control de tipo secuencia, para realizar la declaración de variables de diferentes tipos de datos, así como efectuar llamadas a funciones externas de entrada y salida para asignar y mostrar valores de variables y expresiones.

**Desarrollo de la práctica:**

Durante esta práctica trabajamos con el lenguaje de programación C y la sintaxis utilizada para llevar a cabo la codificación de un programa en este lenguaje, se revisaron las estructuras de control de tipo secuencial para declarar variables, hacer operaciones e imprimir en pantalla.

Sabemos que para resolver un problema usando software lo primero que hay que hacer es el análisis donde se identifican los datos de entrada, datos de salida y restricciones; luego pasamos al diseño de un algoritmo que resuelva el problema de manera eficiente, este puede estar representado de manera gráfica o escrita y después de esto podemos proceder a la etapa de codificación.

**Lenguaje de programación C**

Antes de tener el nombre que ahora lleva, algunos otros lenguajes de programación tuvieron influencia sobre C, primero el lenguaje BCPL, que fue desarrollado por Martin Richards; este influenció en la creación del lenguaje B inventado por Ken Thompson que a su vez llevo al desarrollo e implementación del lenguaje C por Dennis Ritchie.

C es un lenguaje de programación estructurado que ofrece como ventajas economía de expresión, control de flujo y estructuras de datos y un conjunto de operadores. El teorema del programa estructurado, demostrado por Böhm-Jacopini, dicta que todo programa puede desarrollarse utilizando únicamente 3 instrucciones de control: secuencia, selección e iteración.

Para crear un programa en C se deben seguir tres etapas: edición, compilación y ejecución, donde en la edición se escribe el código fuente en lenguaje C; luego, en la compilación a partir del código fuente se genera un archivo ejecutable siempre y cuando la edición se haya realizado correctamente y no contenga errores que impidan su compilación; pasamos entonces a la ejecución donde el archivo creado en la compilación pasa a ejecutarse y a realizar sus funciones.

Un programa en C consiste en una o más funciones, de las cuales una de ellas debe llamarse y es la principal. Al momento de ejecutar un programa objeto, se ejecutarán únicamente las instrucciones que estén definidas dentro de la función principal. La función principal puede contener sentencias, estructuras de control y comentarios. Dentro de las sentencias se encuentran la declaración y asignación de variables, la realización de operaciones básicas, y las llamadas a funciones. Es importante mencionar que al solo ejecutarse las instrucciones que están dentro de la función principal, no se ejecutaran las instrucciones contenidas en las funciones a menos que se manden a llamar durante la edición del código.

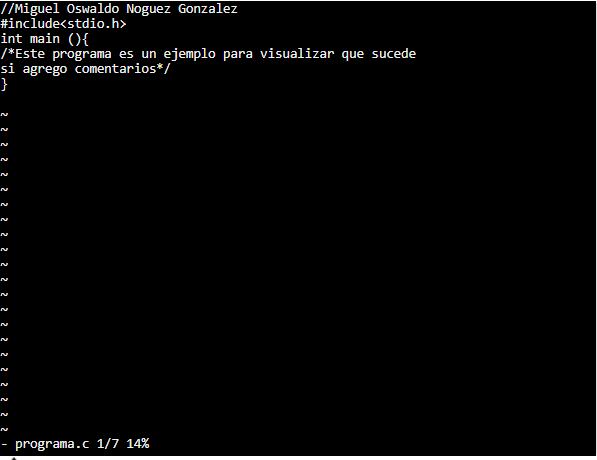
**Comentarios**

Con el objetivo de entender lo que realiza el programa y ser de ayuda para aclarar ciertos aspectos del código están los comentarios, que no afectarán en la salida del programa, solo están visibles en el código; en C existen dos tipos de comentarios que son comentario por línea y cometarios por bloque.

El comentario por línea inicia cuando se insertan los símbolos “//” y termina hasta donde acaba el renglón.

El comentario por bloque inicia con los símbolos “/\*” y termina con “\*/”, este tipo de comentario puede abarcar varios renglones.

A continuación, se muestra un ejemplo del uso de los comentarios y la prueba de que no afectan en la salida del programa. Se creó el programa de nombre desde la terminal de Linux y se usó el editor de texto VI:



Compilación del programa



Ejecución



Podemos ver como al ejecutar al programa no se muestra ninguna salida, ya que no se indica en el código, y aunque hay comentarios estos no son visibles al ejecutar por lo que no afectan en nada al programa.

**Declaración de variables**

Para declarar variables en C se sigue la siguiente sintaxis:

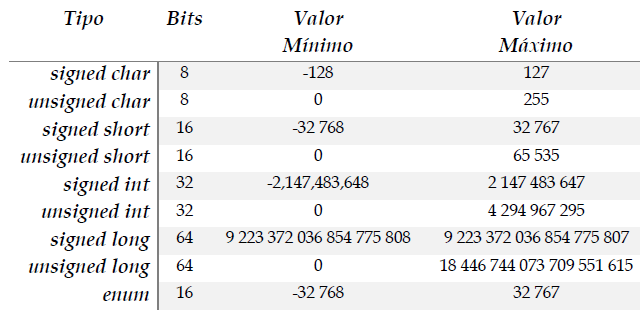
Por lo tanto, una variable puede tener modificadores (opcionales), se debe declarar el tipo de dato que puede contener la variable, se debe declarar el identificador (nombre de la variable) con el que se va a manejar el valor y se puede asignar un valor inicial a la variable. También es posible declarar varios identificadores de un mismo tipo de dato e inicializarlos en el mismo renglón, separando las variables por comas:

Los tipos de datos básicos en C son:

* Caracteres: codificación definida por la máquina.
* Enteros: números sin punto decimal.
* Flotantes: números reales de precisión normal.
* Dobles: números reales de doble precisión.

Variables enteras en C:

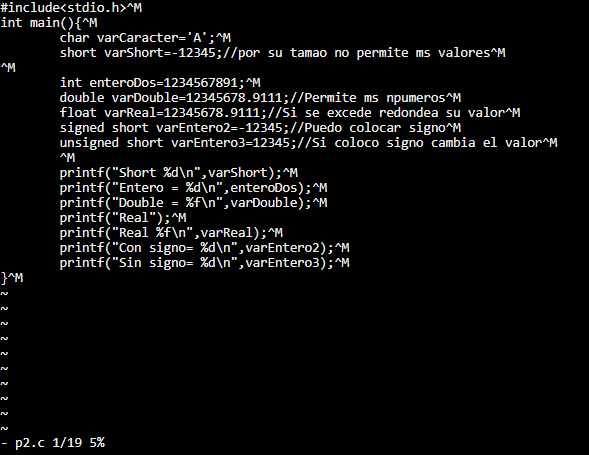
A continuación, se muestran algunas tablas que fueron visualizadas durante la práctica para explicar estos temas:



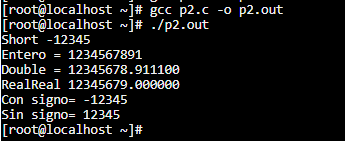
Como podemos ver, las variables pueden llevar o no signo, y de esto dependerá de cual pueda llegar a ser su valor máximo y su valor mínimo.

Si se omite el clasificador, por defecto se considera 'signed'.

A continuación, se muestra un código donde podemos observar como funcionan estos tipos de datos y que pasa si no se especifica el signo, o si ponemos un valor que excede al valor máximo:

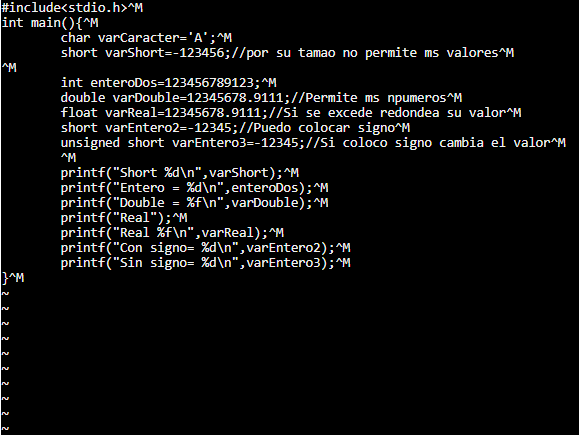


Código del programa

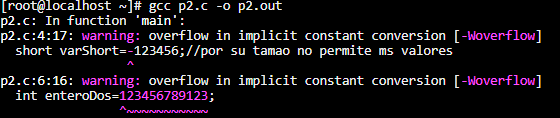


Ejecución del programa

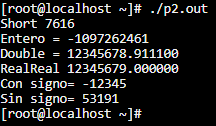
Después de visualizar la salida del código para ver cómo funcionaba el programa hicimos algunas modificaciones con el fin de comprobar cuáles eran las consecuencias de agregar valores más grandes de los que podía aceptar cierto tipo de dato o que ocurría al cambiar el signo de los tipos de dato.



Código del programa modificado

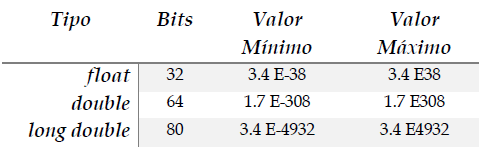


Notamos que al intentar compilar nos arroja dos warnings o advertencias que nos dicen que el tipo de dato short e int contiene un valor que excede su capacidad, es importante resaltar que estas advertencias no impiden que se genere el archivo ejecutable, así que la salida es la siguiente:



Podemos notar que al imprimir las variables se cambian completamente los valores que asignamos en el código, esto es porque exceden la capacidad máxima que pueden almacenar estas variables. En la parte de los signos vemos primero que al imprimir la variable nos regresa el mismo valor que antes de modificar el código a pesar de haber cambiado el tipo de dato de a ; esto es porque cuano no especificamos si llevará signo o no, por defecto se entiende que si lo lleva, por otro lado, cuando se imprime la variable si cambia la salida, y esto es porque agregamos un signo negativo a esta variable que era de tipo , por esto es que nos regresa un valor totalmente diferente. Por lo tanto, hay que tener siempre cuidado al inicializar variables y elegir el tipo de dato a utilizar.

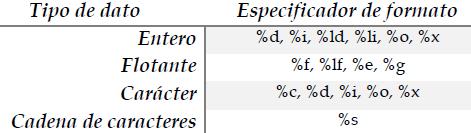
Las variables reales que existen en lenguaje C son:



Es importante mencionar que las variables reales siempre poseen signo.

Para poder acceder al valor de una variable se requiere especificar el tipo de dato.

Los especificadores para los diferentes tipos de datos son:



Este especificador de dato se usa para guardar o imprimir una variable. Por ejemplo, para imprimir un valor entero en base diez se puede usar %d o %i, para enteros largos se usa %ld o %li, por otro lado, para imprimir un entero en base 8 usamos %o y en base 16 se usa %x.

Un valor real se imprime usando %f, para reales de precisión doble se usa %lf, %e para notación científica y %g redondea la parte fraccionaria a 3 dígitos significativos.

Una variable de tipo carácter se imprime usando %c, con los especificadores %d, %i, %o, y %x imprimimos el valor del código ASCII del carácter en base 10, 8 o 16 respectivamente.

Podemos imprimir también cadenas de caracteres y estas se imprimen con el especificador %s.

**Identificador**

Un identificador es el nombre con el que se va a almacenar en memoria un tipo de dato, debe iniciar con una letra, para su nombre se pueden utilizar letras mayúsculas y minúsculas, números y el carácter guion bajo. Es recomendable usar la notación de camello para nombrar estos identificadores o variables.

Al programar en C podemos usar diferentes bibliotecas y comandos para desarrollar nuestro código y que funcione correctamente. Por ejemplo, la biblioteca contiene diversas funciones tanto para imprimir la salida como para leer la entrada.

es una función para imprimir con formato, es decir, se tiene que especificar entre comillas el tipo de dato que se desea imprimir, también se puede combinar la impresión de un texto predeterminado. Ejemplo:

es una función que sirve para leer datos de la entrada estándar (teclado), para ello únicamente se especifica el tipo de dato que se desea leer entre comillas y en qué variable se quiere almacenar. Al nombre de la variable le antecede un ampersand (&), esto indica que el dato recibido se guardará en la localidad de memoria asignada a esa variable. Ejemplo:

Para imprimir con formato también se utilizan algunas secuencias de caracteres , C maneja los siguientes:

* carácter de alarma
* retroceso
* avance de hoja
* salto de línea
* regreso de carro
* tabulador horizontal
* tabulador vertical
* carácter nulo

**Diferencia entre declarar e inicializar variables**

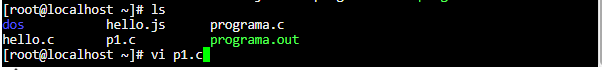
Es importante destacar que existe una gran diferencia entre declarar e iniciar una variable, pues al declarar una variable se le asigna un nombre y tipo de dato, mientras que cuando se inicializa la variable es cuando se le asigna un valor a esta. Ejemplo:

Declaración de la variable.

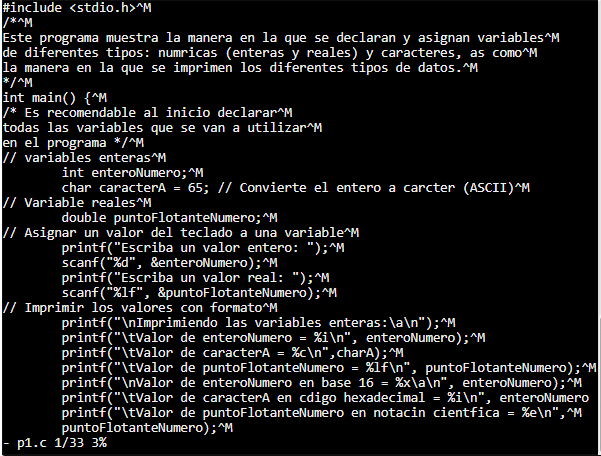
Inicialización de la variable.

**Ejemplo de código que almacena e imprime variables**

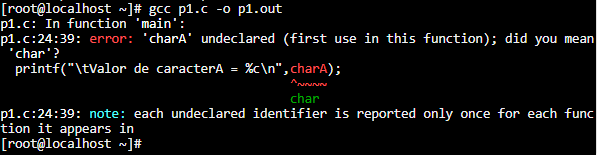
Durante la práctica se visualizó un ejemplo de un código que almacena e imprime variables, este programa fue subido a la terminal de Linux para visualizarlo y ejecutarlo.



Comprobación de que el programa se subió (nombre: p1.c)

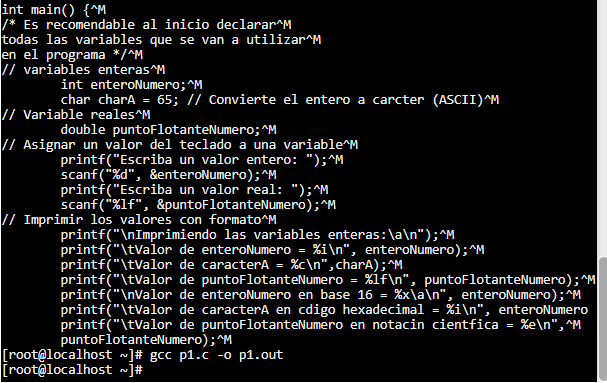


Código del programa

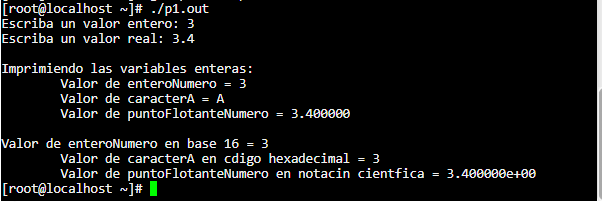


Error en el código al tratar de compilar

Podemos ver como al intentar compilar el programa desde la consola nos sale un mensaje de error, pues en el código se manda a llamar a un variable llamada que no está declarada y por lo tanto no puede ser utilizada. A continuación, se muestra la corrección del problema:



Podemos observar también que se volvió a intentar compilar el programa una vez corregido y ahora si se generó el archivo ejecutable, ya que el código ya no contenía errores.



Ejecución del programa y salida

**Modificadores de alcance**

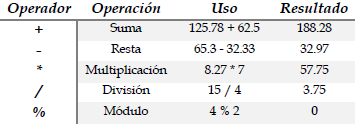
Son modificadores que se pueden agregar al inicio de la declaración de variables, son y .

El modificador impide que una variable cambie su valor durante la ejecución del programa, por convención las constantes se escriben con mayúsculas y se deben inicializar al momento de declararse. Ejemplo:

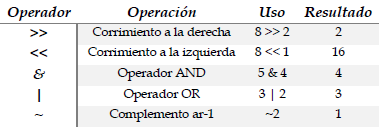
El modificador indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa, es decir, permanece estática en la memoria. Ejemplo:

**Operadores**

Los operadores aritméticos en C son los siguientes:

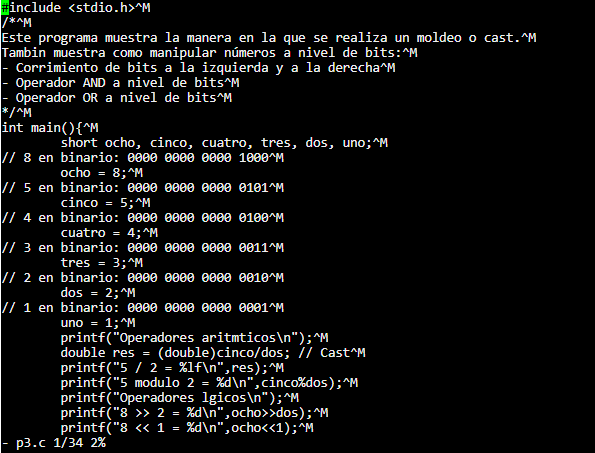


Los operadores lógicos en C son los siguientes:

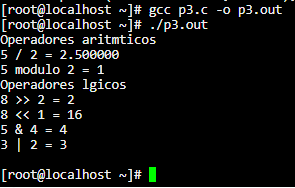


A continuación, se muestra un ejemplo de código donde se usan estos operadores, para poder visualizar como es que funcionan:

Código



Ejecución del programa

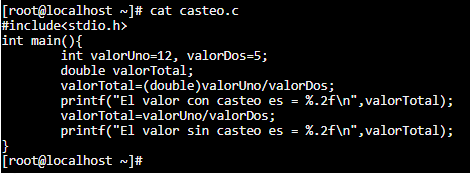


**Moldeo o casteo**

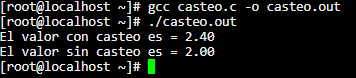
El resultado de una operación entre dos tipos de datos iguales puede dar como resultado un tipo de dato diferente al tipo de dato asignado a una variable inicialmente, en esos casos, para que no existen errores, es necesario moldear el resultado. A este proceso se le conoce como cast o casteo.

Ejemplo de código usando el casteo:

Código



Salida

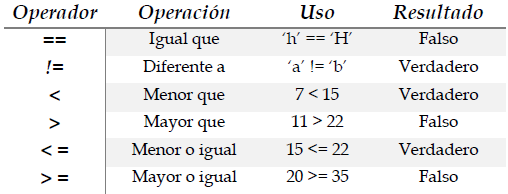


Podemos notar como al aplicar el casteo debemos definir el tipo de dato del resultado de la división, que en este caso es double, vemos como al no hacer esto nos regresa como resultado un número diferente redondeado, que no es el valor de la división, por lo tanto, debemos estar siempre atentos a este tipo de errores y aplicar el casteo cuando sea necesario.

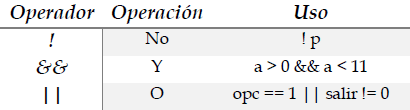
**Expresiones lógicas**

Las expresiones lógicas están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores lógicos. Una expresión lógica puede tomar únicamente los valores verdadero o falso.

Los operadores de relación permiten comparar elementos numéricos, alfanuméricos, constantes o variables y algunos ejemplos son:



Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples:

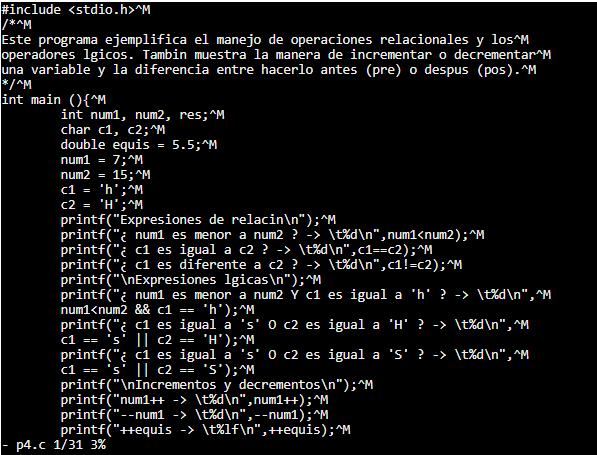


En C podemos ocupar operadores para realizar incrementos y decrementos a un número.

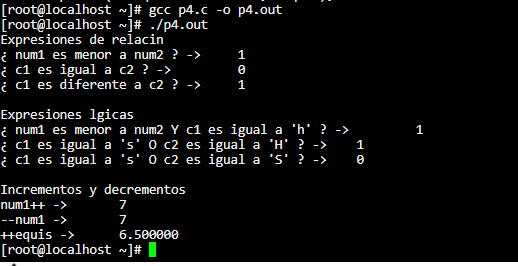
El operador ++ agrega una unidad a su operando. Es posible manejar preincrementos o posincrementos El operador -- resta una unidad a su operando. Se pueden manejar predecrementos o posdecrementos .

A continuación, se muestra un ejemplo en código para poder entender mejor su funcionamiento.

Código



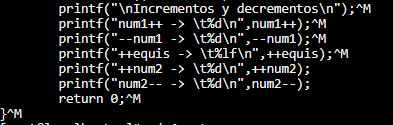
Ejecución



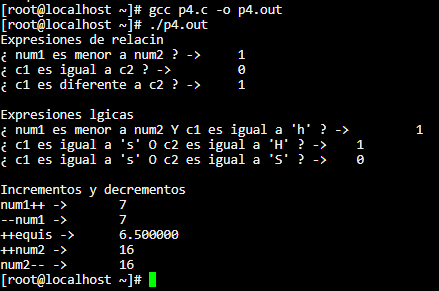
Podemos ver que al usar operadores lógicos la salida siempre es un uno o un cero, este se debe a que en el lenguaje C se manejan los resultados booleanos (Verdadero o falso) con números enteros, cuando el resultado de una comparación es falso el valor regresado es cero, cuando la comparación es verdadera el valor regresado es 1.

Se hicieron también unas modificaciones al código con el objetivo de ver cómo funcionaba el preincremento, posdecremento, predecremento y posincremento.

Código modificado



Ejecución



Para explicar el funcionamiento, es importante saber los valores con los que se inicializaron estas variables, la variable vale 7, vale 15 y vale 5.5, a partir de esto podemos ver que al hacer un posincrementos a se imprime el valor 7, es decir, no cambia; esto es porque el incremento se hace *después* de imprimir la variable, una vez impresa el valor de la variable pasa a ser 8 sin embargo en la siguiente línea se realiza un predecrementos, lo que quiere decir que se decrementa su valor en una unidad antes de ser impresa, recordemos que su valor era 8, entonces pasa a ser 7 de nuevo y se imprime un 7.

Con la variable se hace un preincremento, lo que quiere decir que antes de imprimirse el valor de la variable se incrementa en una unidad su valor, por lo que la salida es un 6.5.

Por otro lado, respecto a la variable se hace un preincremento y por lo tanto el valor que imprime es 16, luego se le aplica un posdecremento cuando su valor ya es 16, es decir primero se imprime el valor de 16 y luego se hace el decremento por lo que ahora el valor de vuelve a ser 15, sin embargo, el valor que se imprime es 16, ya que el decremento se hace después.

**Depuración**

Cuando un programa falla y la información enviada por el compilador es muy general, se puede ejecutar el programa en un contexto controlado para saber, exactamente, dónde está fallando. Puede para corregir nuestro código o para optimizarlo también.

**Ejercicios de tarea:**

**¿Cuál es el dato por default en el lenguaje C? (signed o unsigned)**

Si se omite el clasificador, por defecto el tipo de dato se considera 'signed'. Por esta razón el modificador signed casi no se utiliza, ya que, como se mencionó es el modificador por defecto, sin embargo, es importante tener claro que al declarar variables unsigned permitimos aumentar el valor máximo a almacenar en estas variables.

**¿Qué sucede cuando en una variable tipo carácter se emplea el formato %d, %i, %o y %x?**

Si una variable tipo carácter se imprime con el especificador %d o %i se imprime el valor del código ASCII del carácter en base 10, con el especificador %o se imprime el valor del código ASCII del carácter en base 8 y con el especificador %x se imprime el valor del código ASCII del carácter en base 16.

**Características con las que debe crearse una variable**

Una variable, es un nombre para identificar una o varias posiciones de memoria donde el programa guarda los distintos valores de una misma entidad. Un programa debe definir todas las variables que utilizará antes de comenzar a usarlas, y para definirlas es necesario siempre indicar cuál será el tipo de dato que puede almacenar la variable a fin de indicarle al compilador cuánta memoria debe destinar para albergarla, además podemos inicializar o no las variables, el no inicializarla no provocará que el programa no se pueda ejecutar al menos en el lenguaje C, sin embargo, puede que la salida no sea la deseada, ya que se le asignará un valor por defecto, por lo que, siempre es recomendable inicializar nuestras variables, sobre todo si las usaremos para realizar operaciones. También podemos usar el comando scanf para que el usuario ingrese el valor de la variable, en estos casos no es necesario inicializarla antes, pues lo hará el usuario.

**¿Cuál es la diferencia entre variable estática y constante?**

Una variable constante siempre permanecerá igual durante toda la vida de un programa, no se puede modificar después de definir y no se puede cambiar en todo el programa. La constante con un valor fijo le dice al compilador que evite que el programador la modifique. Cada vez que intente cambiarlo, arrojará un mensaje de error.

Por otro lado, una variable estática indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa, es decir, permanece estática en la memoria. Para explicar su funcionamiento es preciso entender el concepto de clase, que es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables y su comportamiento. Entonces, la variable estática es una propiedad de una clase en lugar de la instancia de la clase. Se almacena en el área de memoria del segmento de datos y el mismo valor se comparte con todas las instancias de esa clase. Se puede asignar para tipos de referencia y establecer en tiempo de ejecución. El modificador estático representa una especie de valor global para todas las instancias de esa clase y puede llamarlas usando el nombre de la clase.

**En qué momento se emplean los dos tipos de diferentes ( < > != )**

Estos tipos de condiciones lógicas indican “desigualdad” (no hay diferencia entre estos símbolos, indican lo mismo), sirven para hacer una prueba de igualdad entre dos variables o valores en el programa; el resultado de esta evaluación son los valores booleanos verdadero y falso, podemos emplear estos símbolos para evaluar condiciones y así poder realizar ciertas acciones u otras en el desarrollo de un programa.

**Crea un programa en el que declares 4 variables haciendo uso de las reglas signed/unsigned, las cuatro variables deben ser solicitadas al usuario (se emplea scanf) y deben mostrarse en pantalla (emplear printf)**

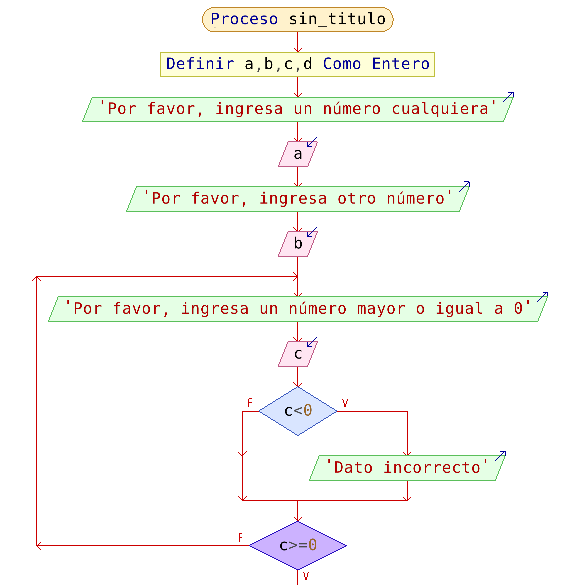
ANÁLISIS

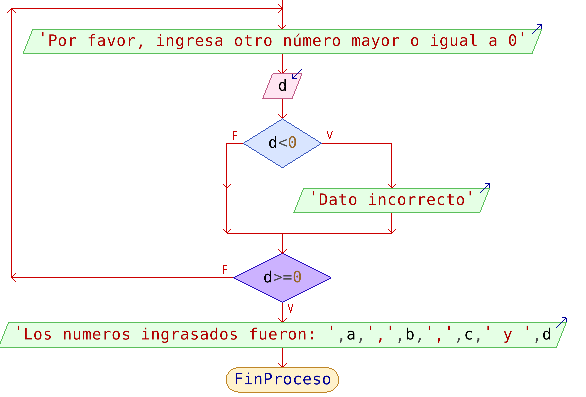
Datos de entrada: Cuatro variables (a,b,c y d) las cuales su valor será introducido por el usuario.

Datos de salida: Impresión de las variables las cuales el usuario indicó su valor.

Restricciones: Las variables a y b serán variables enteras con signo y las variables c y d serán variables sin signo por lo que no se aceptarán números negativos para estas.

DIAGRAMA DE FLUJO





/\*Debido a que en PSEint no se puede especificar si lleva signo o no la variable, se declararon simplemente como enteras, sin embargo, sí se impide que el usuario introduzca variables negativas para c y d que se especificaron en el análisis como variables sin signo\*/

PSEUDOCÓDIGO

INICIO

a,b,c,d:ENTERO

ESCRIBIR "Por favor, ingresa un número cualquiera"

LEER a

ESCRIBIR "Por favor, ingresa otro número"

LEER b

HACER

ESCRIBIR "Por favor, ingresa un número mayor o igual a 0"

LEER c

SI c< 0 ENTONCES

ESCRIBIR "Dato incorrecto"

FINSI

MIENTRAS c<0

HACER

ESCRIBIR "Por favor, ingresa otro número mayor o igual a 0"

LEER d

SI d < 0 ENTONCES

ESCRIBIR "Dato incorrecto"

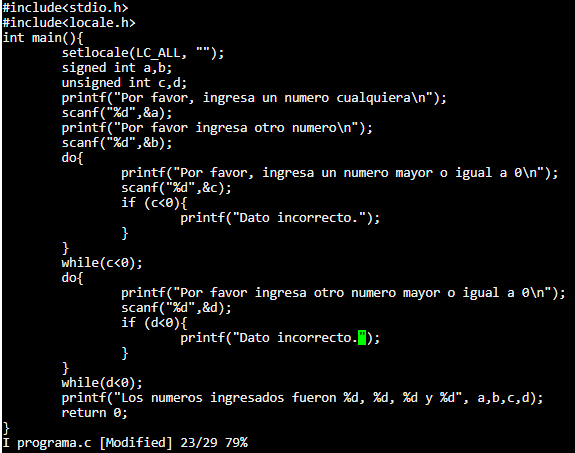
FINSI

MIENTRAS d<0

ESCRIBIR "Los números ingresados fueron: " a "," b "," c " y " d

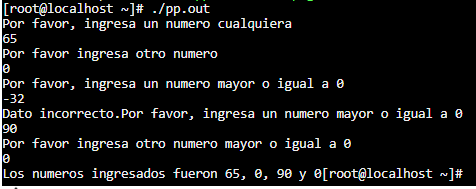
FIN

CODIFICACIÓN

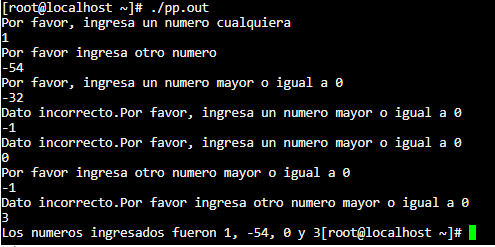


PRUEBAS DE ESCRITORIO

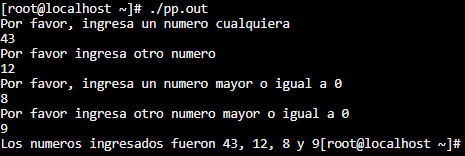
1.



2.



3.



/\*Podemos ver que el programa no deja introducir número negativos debido a que la variable es *unsigned*, si quitáramos esta restricción el valor que se imprimiría seria incorrecto.\*/

**Crea un programa que le solicite su edad al usuario, leer datos (emplear scanf) y mostrarlo en pantalla**

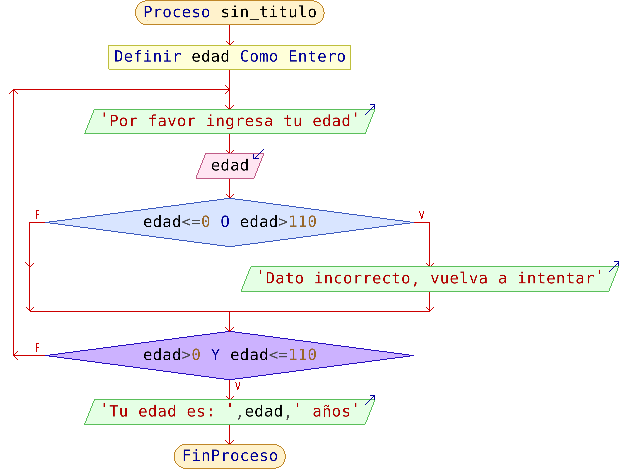
ANÁLISIS

Datos de entrada: Edad del usuario, que será almacenada en la variable de tipo entero *edad*.

Datos de salida: Impresión en pantalla de la edad del usuario.

Restricciones: La edad ingresada no puede ser menor o igual que cero ni tampoco puede ser mayor que 110

DIAGRAMA DE FLUJO



PSEUDOCÓDIGO

INICIO

edad:ENTERO

HACER

ESCRIBIR "Por favor ingresa tu edad"

LEER edad

SI edad<= 0 o edad >110 ENTONCES

ESCRIBIR "Dato incorrecto, vuelva a intentar"

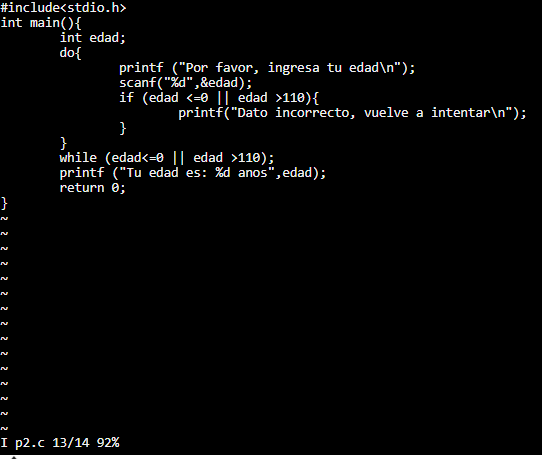
FINSI

MIENTRAS edad<= 0 o edad >110

ESCRIBIR "Tu edad es: " edad " años"

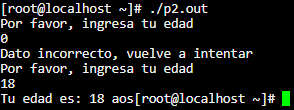
FIN

CODIFICACIÓN

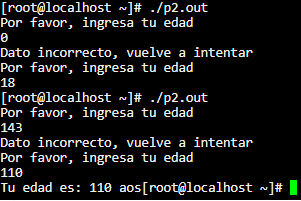


PRUEBAS DE ESCRITORIO

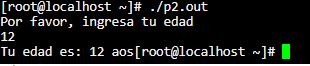
1.



2.



3.



**Revisar y colocar cuando se emplea MOD y cuando se emplea % (pseudocódigo y códificación) agrega un ejemplo de su uso**

La palabra reservada MOD y el símbolo % son sinónimos. Este operador se utiliza en las expresiones aritméticas entre dos operandos, para obtener el residuo del primero dividido por el segundo. Por ejemplo 7 MOD 3 devolverá 1, ya que al realizar la división entera de 7 entre 3 se obtiene 2 como cociente y 1 como residuo.

Ejemplo en un programa:

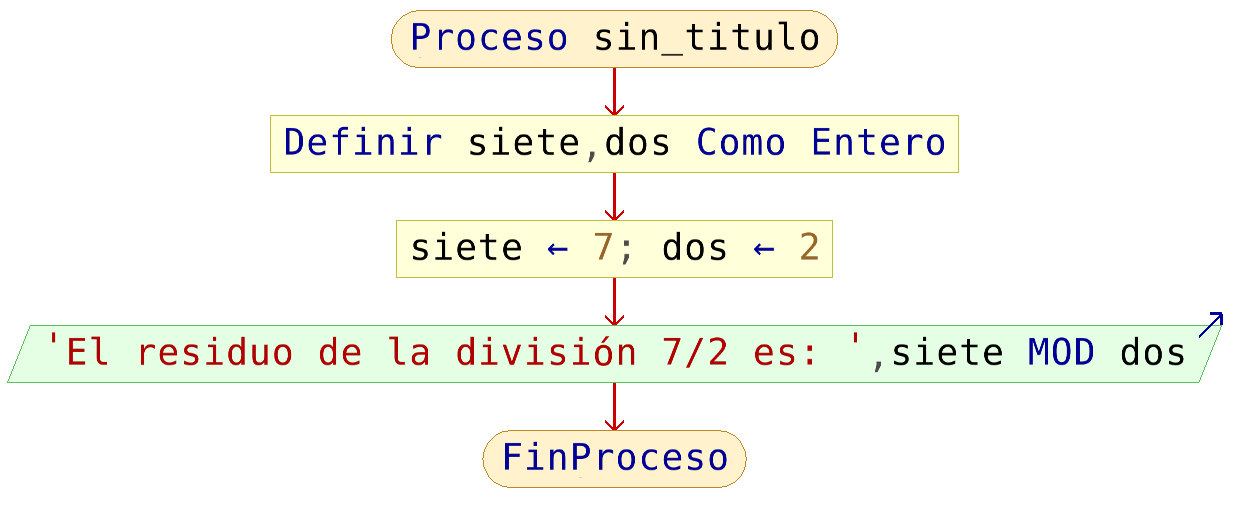
ANÁLISIS

Datos de entrada: No habrá datos introducidos por el usuario solo se crearán las variables *siete* y *dos*.

Datos de salida: Residuo de dividir 7 entre 2.

Restricciones: No hay restricciones, ya que las variables serán inicializadas por nosotros, la única restricción que podríamos considerar es que las variables creadas deberán ser de tipo entero, ya que para trabajar con MOD los operandos debe ser enteros.

DIAGRAMA DE FLUJO



PSEUDOCÓDIGO

INICIO

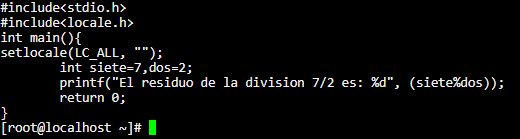
siete, dos:ENTERO

siete :=7; dos :=2

ESCRIBIR "El residuo de la división 7/2 es: " siete MOD dos

FIN

CODIFICACIÓN

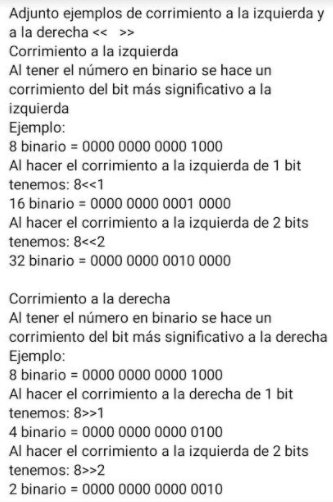


PRUEBAS DE ESCRITORIO



**Da tu opinión sobre cada uno de los ejemplos realizados en la imagen.**

1.



COMENTARIOS:

De este ejercicio se puede concluir que hacer corrimientos hacia la izquierda o hacia a la derecha funciona con número binarios y el corrimiento se hace dependiendo de la posición del bit más significativo que en estos casos es el 1, ya que los demás son ceros. Lo que indica esta instrucción es hacer el corrimiento de n posiciones hacia la derecha o hacia la izquierda del bit más significativo de un número en binario; podríamos decir que una fórmula para entenderlo sería:

numeroEnBinario<<corrimientoDeNPosicionesALaIzquierda

numeroEnBinario>>corrimientoDeNPosicionesALaIzquierda

Podemos notar también que cuando hacemos corrimientos a la derecha el número se hace más pequeño debido a que es binario y cuando hacemos corrimientos a la izquierda se hace más grande.

2. Programa que explica AND y OR a nivel bits

#include <stdio.h>

/\*

 Este programa muestra como manipular números a nivel de bits:

 - Operador AND a nivel de bits

 - Operador OR a nivel de bits

\*/

int main(){

 short cinco, cuatro, tres, dos;

 printf("Operadores aritméticos\n");

 //OPERACIÓN AND solo en donde coincide 1 se coloca 1 en resultado

   unsigned int a = 29;    /\* 29 = 0001 1101 \*/

   unsigned int b = 48;    /\* 48 = 0011 0000 \*/

   //                             \_\_\_\_\_\_\_\_\_\_\_

   // 32 =   0001 0000

   cinco = 5; //5 en binario: 0000 0000 0000 0101

   cuatro = 4;//4 en binario: 0000 0000 0000 0100

 //                           \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 //                        4= 0000 0000 0000 0100

 printf("29 & 48 = %d\n", a&b );

 printf("5 & 4 = %d\n",cinco&cuatro);

 printf("4 & 3 = %d\n",cuatro&tres);

//OPERACIÓN OR  todos los 1 se colocan en el resultado

 tres = 3;// 3 en binario: 0000 0000 0000 0011

 dos = 2; /\* 2 en binario: 0000 0000 0000 0010

                           \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

             3= 0000 0000 0000 0011

5 en binario: 0000 0000 0000 0101

4 en binario: 0000 0000 0000 0100

              \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

        5=    0000 0000 0000 0101

 29 = 0001 1101

 48 = 0011 0000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 61 = 0011 1101 \*/

 printf("3 | 2 = %d\n",tres|dos);

 printf("5 | 4 = %d\n",cinco|cuatro);

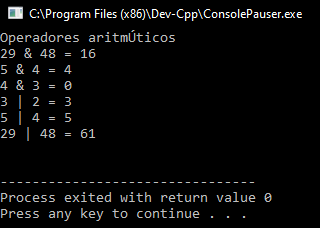
printf("29 | 48 = %d\n", a|b );

 printf("\n");

 return 0;

}

SALIDA:



COMENTARIOS:

Con este código podemos ver como modificar números a nivel de bits en binario con los operadores & (AND) y | (OR), donde el primero nos indicará en que posiciones coincide un bit encendido (“1”) al comparar dos números en binario, y en las posiciones donde coincida se conservará ese “1” en el resultado, por ejemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| Número en decimal | Número en binario | Posiciones donde coincide (binario) | Resultado en decimal: |
| 9 | 0000 **1**00**1** | 0000 1001 | 9 |
| 29 | 0001 **1**10**1** |

Por lo tanto, el resultado en decimal de la operación es:

Por otro lado, El operador | hace algo muy parecido, solo que, en lugar de conservar en el resultado solo las posiciones donde coincide el bit encendido, conserva todas las posiciones donde haya un bit encendido. Ejemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| Número en decimal | Número en binario | Posiciones donde hay un bit encendido | Resultado en decimal: |
| 9 | 0000 **1**00**1** | 0001 1101 | 29 |
| 29 | 000**1** **11**0**1** |

Por lo tanto, el resultado en decimal de la operación es:

**Conclusiones:**

Con el desarrollo de esta práctica nos podemos dar cuenta de la importancia de conocer el entorno y componentes necesarios para llevar a cabo la codificación de un programa en el lenguaje de programación C, por lo tanto, es importante conocer la sintaxis del lenguaje, como es que las variables se declaran y como se inicializan, además de conocer los tipos de datos y restricciones que estas tienen, ya que, como hemos visto con los ejemplos dados durante el desarrollo, el no poner atención a este tipo de restricciones o el no especificar el especificador de formato correctamente puede provocar que nuestro código se ejecute, pero contenga errores y no realice exactamente la función que nosotros queremos.

Además, es fundamental también, conocer los tipos de operadores aritméticos y lógicos que podemos usar en nuestros programas, ya que estos nos ayudarán a validar datos o hacer cualquier tipo de operación con nuestras variables, por lo que conocerlos es importante para encontrar una solución óptima para el problema que se quiera resolver.

Por otro lado, llevar a cabo pruebas de escritorio después de realizar nuestro código nos puede ayudar a corregir problemas y optimizar nuestros programas, a este proceso lo podemos conocer también como depuración; de esta forma estaríamos evitando cualquier tipo de imprevisto y nos aseguraremos de que el programa cumpla con la solución. En caso de encontrar cualquier error en las pruebas, es necesario regresar a la edición del código para identificar que es lo que está fallando o que es lo que no nos está gustando de nuestro programa, siguiendo estos pasos es como conseguiremos desarrollar programas funcionales.

**Bibliografía:**

* Facultad de Ingeniería UNAM. (s. f.). Laboratorio Salas A y B. Recuperado 20 de noviembre de 2020, de <http://lcp02.fi-b.unam.mx/>
* Las variables y los tipos básicos en C. (s. f.). Recuperado 23 de noviembre de 2020, de <http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/variables.htm>
* Inicialización de variables - Programación 2012-2013. (s. f.). Recuperado 23 de noviembre de 2020, de <https://sites.google.com/site/pro012iessanandres/java/inicializacion-de-variables>
* ¿Cuál es la diferencia entre variable estática y constante? (s. f.). Recuperado 23 de noviembre de 2020, de <https://svcministry.org/es/dictionary/what-s-the-difference-between-static-and-constant-variable/>
* Clase (informática). (2020, 5 noviembre). Recuperado 23 de noviembre de 2020, de <https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)#:%7E:text=En%20inform%C3%A1tica%2C%20una%20clase%20es,los%20sustantivos%20en%20el%20lenguaje.&text=La%20programaci%C3%B3n%20orientada%20a%20objetos,para%20los%20tipos%20de%20objetos%20>.
* Oracle Help Center. (s. f.). Comparison Conditions. Recuperado 23 de noviembre de 2020, de <https://docs.oracle.com/database/121/SQLRF/conditions002.htm#CJAGAABC>
* MOD %. (s. f.). Recuperado 23 de noviembre de 2020, de <http://cdiv.sourceforge.net/cdivhlp/1048.htm#:%7E:text=La%20palabra%20reservada%20MOD%20y,cociente%20y%201%20como%20resto>.