Pontificia Universidad Católica del Perú

Estudios Generales Ciencias

1INF01 - Fundamentos de Programación

Guía del laboratorio peliminar
Estructura básica de un programa

21 de agosto de 2022

Índice general

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.0	17.08.2018	A.Melgar	Versión inicial.
1.1	26.03.2019	A.Melgar	Se incrementaron la cantidad de problemas propuestos y se añadió color al código en ANSI C.
1.1	30.04.2019	L. Hirsh	Revisión de la versión 1.1.
1.2	18.06.2019	A.Melgar	Revisión de la redacción. Se incrementaron la cantidad de problemas propuestos.
1.2	30.07.2019	J. Zárate	Revisión de la versión 1.2.
1.2	30.07.2019	A.Melgar	Corrección de la revisión 1.2 realizada por la profesora Jénnifer Zárate.
1.3	09.03.2020	A.Melgar	Se incrementaron la cantidad de problemas propuestos y se clasificaron éstos en las categorías Nivel Básico, Nivel Intermedios, Nivel Avanzados.
1.4	29.08.2020	A.Melgar	Se realizó la revisión para el semestre 2020-2, se cambió el nombre del documento a "Guía del laboratorio preliminar" y se incrementaron la cantidad de problemas propuestos.
1.5	17.03.2021	S.Vargas	Se realizó la revisión para el semestre 2021-1, se modificó la redacción y casos de prueba en algunos problemas propuestos.
1.5.1	20.08.2021	S.Vargas	Se realizó la revisión para el semestre 2021-2.
1.5.2	15.03.2022	S.Vargas	Se realizó la revisión para el semestre 2022-1.
1.5.3	21.08.2022	S.Vargas	Se realizó la revisión para el semestre 2022-2.

Siglas

ASCII	American Standard Code for Information Interchange
EEGGCC	Estudios Generales Ciencias
IDE	Entorno de Desarrollo Integrado
PUCP	Pontificia Universidad Católica del Perú
RAE	Real Academia Española

Capítulo 1

Guía del Laboratorio preliminar

1.1. Introducción

Esta guía ha sido diseñada para que sirva como una herramienta de aprendizaje y práctica para el curso de Fundamentos de Programación de los Estudios Generales Ciencias (EEGGCC) en la Pontificia Universidad Católica del Perú (PUCP). En particular, se focaliza en el tema "Estructura básica de un programa".

Se busca que el alumno resuelva paso a paso las indicaciones dadas en esta guía, contribuyendo de esta manera a los objetivos de aprendizaje del curso, en particular, en la aplicación de la lógica proposicional en el diseño de programas imperativos. Al finalizar el desarrollo de esta guía y complementando con lo que se realizará en el correspondiente laboratorio, se espera que el alumno:

- Comprenda la estructura básica de un programa imperativo.
- Aplique los operadores de asignación, aritméticos, relacionales y lógicos en un programa imperativo.
- Implemente programas que utilicen los operadores de asignación, aritméticos, relacionales y lógicos en un lenguaje de programación imperativo.
- Declare correctamente las variables con el tipo de dato que corresponda.

1.2. Materiales y métodos

Como lenguaje de programación imperativo se utilizará el lenguaje C. Como Entorno de Desarrollo Integrado (IDE) para el lenguaje C se utilizará Dev-C++[1]. No obstante, es posible utilizar otros IDEs como Visual Studio Code.

1.3. Estructura básica de un programa

Como ya se mencionó anteriormente, el objetivo de esta guía es comprender la estructura de un programa imperativo. Existen varios paradigmas de programación entre los que se pueden mencionar al: paradigma imperativo, paradigma orientado a objetos, paradigma funcional, paradigma lógico, entre otros.

El paradigma imperativo busca la descripción de programas a través de cambios de estado para lo cual se elaboran sentencias que realizan estos cambios. Imperar, según la Real Academia Española (RAE), significa mandar, ordenar[2]. En la programación imperativa, los programas son conjuntos de órdenes que describen cómo la computadora debe realizar una tarea determinada. Hacer un programa en este paradigma imperativo

[1] http://sourceforge.net/projects/orwelldevcpp

[2] http://dle.rae.es/?id=L2wAyos

significa básicamente escribir una lista de instrucciones que ordenen al computador la ejecución de dichas tareas.

El paradigma de programación define una forma común en la que se deben escribir los programas, pero para escribir programas se deben utilizar lenguajes formales de forma tal que puedan ser "entendibles" por el computador. Es así que surge la necesidad de contar con lenguajes de programación. Un lenguaje de programación es un lenguaje formal que describe la manera en la que se escribirán los programas. Cada lenguaje de programación definirá los símbolos que puede utilizar así como también las reglas sintácticas (estructura) y semánticas (significado).

Cada lenguaje de programación implementa por lo menos un paradigma de programación. Dentro de la programación imperativa se tienen diversos lenguajes de programación entre los que se pueden mencionar: Basic, Fortran, Pascal, C, entre otros. Para este curso se utilizará como lenguaje de programación el C.

El lenguaje C se empezó a desarrollar entre los años 1969–1973 en los laboratorios Bell por Dennis Ritchie[3] basándose en los lenguajes B y BCPL. Entre las características que posee el lenguaje C se puede mencionar que:

- Es un lenguaje estructurado.
- Hace uso extensivo de punteros para manejar la memoria, arreglos, estructuras y funciones.
- Posee construcciones de alto nivel y construcciones de bajo nivel.
- Puede ser compilado en una amplia gama de computadoras.

Posteriormente, entre los años 1983 y 1985, Bjarne Stroupstrup[4], también en los laboratorios Bell desarrolló el lenguaje C++. El C++ es una extensión de C que busca:

- Soportar la abstracción de datos.
- Soportar la programación orientada a objetos.
- Soportar la programación genérica.

En el programa 1.1 se puede apreciar un programa simple que servirá para describir y comprender la estructura básica de un programa en C. Podrá encontrar una tarjeta de referencia del lenguaje C en español en la URL http://arantxa.ii.uam.es/~cantador/slides/tarjeta_referencia-ANSI_C.pdf y una tarjeta de C en inglés en la URL https://users.ece.utexas.edu/~adnan/c-refcard.pdf.

Programa 1.1: Primer programa en C

```c
#include <stdio.h>

/*
 * Este primer programa tiene la intención de
 * presentar la estructura básica de un programa
 * usando el lenguaje C
 */

int main() {
    printf("Bienvenidos al curso de Fundamentos de Programación");
    return 0;
}
```

En el programa 1.1 se puede apreciar las secciones básica de un programa en C: la directivas del preprocesador (`#include`), los comentarios (`/* */`) y la función `main`. A continuación se analizará al detalle cada una de las secciones.

[3]En la URL https://www.bell-labs.com/usr/dmr/www/ podrán encontrar los trabajos realizado por Dennis Ritchie.

[4]En la URL http://www.stroustrup.com/ se encuentra la página Web de Bjarne Stroupstrup la cual contiene muchos recursos relacionados al lenguaje C++.

1.3.1. Directivas del preprocesador

El computador solamente puede procesar instrucciones en lenguaje de máquina. El lenguaje de máquina es un lenguaje que depende del hardware del computador, es decir que es dependiente de la máquina, es por esta razón que se le denomina lenguaje de máquina. El lenguaje de máquina pertenece a la categoría de los lenguajes de bajo nivel que se caracterizan por la complejidad al momento de escribir programas.

Escribir un programa usando lenguaje de bajo nivel es una tarea compleja pues antes que nada hay que conocer el hardware en donde se ejecutará el programa, dependiendo del hardware existirá un conjunto de instrucciones que se podrá utilizar. Si se desease ejecutar el mismo programa en otro computador, se tendría que escribir otro programa con las instrucciones que el hardware del otro computador reconociese.

En la figura 1.1 se puede apreciar un ejemplo en lenguaje de máquina de un programa que busca sumar los números 1234 y 4321. Como se puede apreciar el programa está escrito en binario, es decir mediante secuencias de unos y ceros. Esto hace que este tipo de programas sean difíciles de entender para el ojo humano.

TABLE 4-3
A *machine code* program for adding 1234 and 4321. This is the lowest level of programming: direct manipulation of the digital electronics. (The right column is a continuation of the left column).

10111001	00000000
11010010	10100001
00000100	00000000
10001001	00000000
00001110	10001011
00000000	00011110
00000000	00000010
10111001	00000000
11100001	00000011
00010000	11000011
10001001	10100011
00001110	00000100
00000010	00000000

Figura 1.1: Ejemplo de un programa de máquina. Imagen tomada del libro [1].

Debido a esta dependencia (del programa con el hardware) y además a la complejidad del entendimiento para el humano del lenguaje de bajo nivel, surgen los denominados lenguajes de alto nivel. Los programas escritos usando lenguajes de alto nivel son fáciles de comprender para el ojo humano. El lenguaje C puede ser considerado como un lenguaje de alto nivel.

El problema radica en que los computadores solo pueden procesar lenguaje de máquina. Entonces surge la pregunta: ¿cómo hacer para escribir programas en lenguajes de alto nivel y que estos puedan ser procesados por las máquinas? Para conseguir esto, es necesario que exista una herramienta que permita traducir el lenguaje de alto nivel a lenguaje de bajo nivel. Es así que nacen los traductores.

Existen dos tipos de traductores, los compilares y los intérpretes. Los compiladores sustituyen cada instrucción del programa por una sucesión equivalente de instrucciones en lenguajes de máquina. En la compilación la traducción se hace en un único proceso. En la interpretación la traducción se hace conforme se va ejecutando el programa, línea por línea.

El lenguaje C es un lenguaje que se compila, por lo tanto los IDE que implementan este lenguaje incorporan un compilador. El preprocesador es la parte del compilador que realiza la primera etapa de traducción previa a la compilación. En esta etapa de preprocesamiento se ejecutan las directivas del preprocesador. Estas directivas son fáciles de reconocer pues empiezan con el símbolo numeral (#). En el programa 1.1 se puede apreciar el uso de directivas del preprocesador en la línea 1: `#include <stdio.h>`.

Las directivas del preprocesador poseen varios usos, entre ellos:

- Permitir la inclusión de archivos (`#include`).
- Definir macros y constantes simbólicas (`#define`).
- Soportar la compilación condicional (`#ifdef`).

En esta ocasión, detallaremos la directiva de preprocesamiento inclusión de archivos `#include`.

Inclusión de archivos

Un programa en C está compuesto por varios tipos de archivos. Los más comunes suelen tener la extensión .c y .h. Los archivos con extensión .c suelen contener la implementación de rutinas y se le suelen llamar archivos de código fuente. Los archivos con extensión .h se les denomina archivos de cabecera (.h del inglés *header file*). Los archivos de cabecera suele contener declaración de rutinas, tipos de datos, estructuras, variables u otros elementos que facilitan la escritura del código fuente.

En el programa 1.1, la directiva de preprocesamiento `#include <stdio.h>` especifica que se debe incluir el archivo de cabecera `stdio.h`. Este archivo es un archivo estándar de C. Cuando se requiera incluir arhivos estándard de C se le antepone los símbolos < y >.El identificador `stdio.h` es un acrónimo de *__s__tandard __i__nput-__o__utput __h__eader* y contiene, entre otras cosas, las declaraciones de las funciones de la biblioteca estándar de entrada y salida de C.

Si se requiere construir programas en C que lean o escriban datos (ver sección 1.4), como lo serán la mayoría de programas que se verán en este curso, se debe incluir el archivo de cabecera `stdio.h`.

1.3.2. Comentarios

Los comentarios son un elemento opcional en un programa en C, pero es recomendable colocarlos para explicar el objetivo de las rutinas. Esto facilita el entendimiento posterior del programa. En C los comentarios se colocan entre los símbolos `/*` y `*/`. Todo lo que va entre estos símbolos es considerado un comentario, por lo tanto no se compila.

En el programa 1.1 se puede apreciar el ejemplo de uso de un comentario entre las líneas 3 y 7.

1.3.3. La función `main`

La función `main`, como su nombre lo indica, es la función principal en un programa en C. Es la primera función que se ejecuta, por lo que todo programa en C deberá tener una función principal.

En los lenguajes de programación, las funciones se caracterizan por retornar un valor, es por esto que al declarar la función `main` se le indica qué tipo de resultado debe devolver. En la línea 9 del programa 1.1 se puede apreciar cómo se declara la función `main` como `int main()`. `int` es el tipo de dato del resultado que se retornará, en este caso un entero. Los paréntesis vacíos () indican que en la función `main` no se han definido parámetros.

Para delimitar el bloque de instrucciones que se ejecutarán dentro de la función `main` se utilizan las llaves. En C el símbolo { abre un bloque de instrucciones y el símbolo } lo cierra.

¿Cómo hace la función `main` para retornar el valor entero? Cómo se puede observar en la línea 11 del programa 1.1 existe la instrucción `return 0`. Esta instrucción hace que la función `main` retorne el valor de 0. El 0 significa que el programa se ha ejecutado sin problemas. Cuando un programa detecta un problema que no permita su ejecución puede retornar otro valor, por lo general suele ser un código de error. En la función `main`, la instrucción `return` suele ser siempre la última del bloque. Esto suele ser así pues la instrucción `return` además de retornar un valor hace que la ejecución de la función termine.

Dentro del bloque de instrucciones, cada sentencia es finalizada por el símbolo `;` (punto y coma), es por este motivo que luego de las líneas 10 y 11 en el programa 1.1, se encuentra un `;` (punto y coma).

1.4. Salida de datos

Para la salida de datos se utilizará la función `printf`. La función `printf` es una función de la biblioteca estándar de entrada y salida de C que permite imprimir datos según determinado formato en la salida estándar del computador. Como esta función se encuentra en la biblioteca estándar, es necesario invocar al archivo de cabecera que contiene su declaración, en este caso corresponde a `stdio.h`.

1.5. Ejecución del programa

Al ejecutar el programa 1.1 se imprime en la salida estándar el siguiente mensaje:

```
Bienvenidos al curso de Fundamentos de Programación
```

El programa hace esta impresión por causa de la función `printf` de la línea 10. Está función lo que hace es enviar, a la salida estándar, el texto que se encuentra en la cadena de formato (el texto que están entre los símbolos " y "). En el programa en cuestión se imprime `Bienvenidos al curso de Fundamentos de Programación`.

Para poner en práctica

- ¿Qué cambios debería realizar en el programa 1.1 si quisiera imprimir su nombre?
- ¿Qué cambios debería realizar en el programa 1.1 si quisiera imprimir su código de alumno?

El lenguaje C, como muchos lenguajes de programación, tiene una característica importante, es *case sensitive*. Esto significa que es sensible a las mayúsculas y a las minúsculas, por lo que no es lo mismo `main`, `Main` o `MAIN`.

Para analizar

- En el programa 1.1, cambie la línea 9 por `int Main() {` y verifique si el programa compila.
- En el programa 1.1, cambie la línea 10 por `Printf("Hola Mundo");` y verifique si el programa compila.
- En el programa 1.1, cambie la línea 11 por `Return 0;` y verifique si el programa compila.

Luego de realizar las 3 alteraciones propuestas, ¿qué puede decir del lenguaje C en relación a las mayúsculas y minúsculas?

1.6. Variables y tipos de datos

Como ya se mencionó previamente, en el paradigma imperativo los programas se describen mediantes instrucciones que permiten realizar cambios de estados. Pero, ¿cómo se representan estos estados en un programa? En el lenguaje C, los valores de estos estados se almacenan a través de unos elementos denominados variables.

1.6.1. Variables

Una variable se puede definir como un espacio de memoria que almacena un valor de determinado dominio. Al espacio de memoria asociado a determinada variable se le puede referenciar mediante un nombre que en el contexto de los lenguajes de programación se le suele denominar identificador. Una característica distintiva de la variable es que el valor que se almacena en ella puede variar a lo largo de un programa.

Para poder declarar una variable en C se debe indicar a qué dominio pertenecerá el valor que se pretende almacenar en la variable, para esto se hace necesario la utilización de tipos de datos.

1.6.2. Tipos de Datos

El lenguaje C básicamente trabaja con números. Existen varios tipos de datos en C, pero en esta guía nos enfocaremos en 3: `char`, `int` y `double` (ver tabla 1.1).

Tanto el `char` como el `int` almacenan números enteros. La diferencia entre ambos radica en los límites, es decir en el rango de números que se pueden representar con cada tipo de dato. El tipo de dato `char` se almacena en 8 bits (1 byte) por lo que el rango de números que se puede representar con un `char` es $[-128, +127]$. Por otro lado el números de bits que se utiliza para representar un `int`, depende del compilador. En compiladores donde el `int` se representa con 16 bits (2 bytes) el rango de números que se puede representar con este tipo de datos es $[-32,768, +32,767]$. En compiladores donde el `int` se representa con 32 bits (4 bytes) el rango de números que se puede representar con este tipo de datos es $[-2,147,483,648, +2,147,483,647]$.

Tabla 1.1: Tipos de datos en C.

Tipo de dato	Descripción	Formato
char	Número entero que representa un código en la tabla American Standard Code for Information Interchange (ASCII) extendida (tipo de dato más pequeño en C).	%c
int	Número entero.	%d
double	Número real de doble precisión.	%lf

Para poner en práctica

¿Cómo se puede determinar cuál es el valor máximo y mínimo que se puede representar con el tipo de dato `int` en el compilador de C que está utilizando? Siga los siguientes pasos para alterar el programa 1.1 y obtener la respuesta.

- Incluya el archivo de cabecera `limits.h` en la sección de directivas del preprocesador.
- Incluya las siguientes instrucciones inmediatamente después de la línea 10:

```
printf("%d\n", INT_MAX);
printf("%d\n", INT_MIN);
```

El `double`, por su parte, permite la representación de números reales de doble precisión. Se representa mediante 64 bits (8 bytes). Se basa en el estándar IEEE 754 en donde la doble precisión se representa mendiante 1 bit para el signo, 11 bits para el exponente, 53 bits para la mantisa.

Para poner en práctica

¿Cómo se puede determinar cuál es el valor máximo y mínimo que se puede representar con el tipo de dato `double` en el compilador de C que está utilizando? Siga los siguientes pasos para alterar el programa 1.1 y obtener la respuesta.

- Incluya el archivo de cabecera `float.h` en la sección de directivas del preprocesador.
- Incluya las siguientes instrucciones inmediatamente después de la línea 10:

```
printf("%lf\n", -DBL_MAX);
printf("%lf\n", DBL_MAX);
```

En la tabla 1.1 se puede observar, en la tercera columna, el formato asociado al tipo de dato. Estos caracteres del formato servirán para la realización de operaciones de entrada (función `scanf`) y salida de datos (función `printf`).

1.6.3. Declaración de variables en C

Declaración

La declaración permite asignarle a la variable un espacio de memoria para que pueda almacenar el valor de un tipo de dato. La forma general para la declaración de variables es como sigue:

Programa 1.2: Declaración de variables en C

```
tipo de dato lista de identificadores;
```

A la izquierda se coloca el tipo de dato para la variable (`char`, `int`, `double`) seguido de la lista de identificadores. Recuerde que el identificador será el nombre de la variable. En caso exista más de una variable, los identificadores se deben separar por comas (`,`). En el programa 1.3 se puede apreciar unos ejemplos de declaración de variables

Programa 1.3: Ejemplo de declaración de variables en C

```c
int numero, suma;
double promedio;
char sexo;
```

Regla para formar identificadores en C

Como se mencionó anteriormente, el nombre de cada variable se conoce como identificador. La formación de todo identificador en C sigue determinadas reglas. Estas son:

- El primer carácter debe ser siempre una letra del alfabeto inglés (`[a..z]`, `[A..Z]`) o el símbolo de subrayado (`_`).
- Los demás caracteres pueden ser letras del alfabeto inglés (`[a..z]`, `[A..Z]`), dígitos (`[0..9]`) o el símbolo de subrayado (`_`).
- C solo reconoce los primeros 31 caracteres de un identificador.
- No debe ser una palabra clave (ver tabla 1.2).

Tabla 1.2: Palabras clave en C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Recordar que:

El lenguaje C es *case sensitive*, eso quiere decir que es sensible a las mayúsculas y las minúsculas. Por lo tanto, para el lenguaje C, `suma`, `Suma` y `SUMA` serán 3 identificadores diferentes. Se recomienda nombrar a los identificadores con minúsculas.

1.7. Operadores de asignación

En el lenguaje C existen símbolos especiales que permiten realizar operaciones específicas predefinidas usando argumentos (*input*). A estas operaciones se les conoce con el nombre de operadores y pueden ser de diversos tipos, entre ellos se pueden mencionar a los aritméticos (suma, resta, multiplicación, división), relacionales (mayor que, menor que, igual a, diferente de), lógicos (conjunción, disyunción, negación), entre otros. A los argumentos de los operadores se les denomina operandos y son los datos a través de los cuales se ejecuta determinada operación. De esta manera si queremos sumar los números 13 y 4, en el lenguaje C se puede utilizar el operador de suma el cual es implementado con el símbolo `+`. A los números 13 y 4 se les denomina operandos del operador. Los operadores que requieren solamente un operando se les llama unarios mientras que a los operadores que requieren dos operandos se les llama binarios. Los operadores se pueden combinar para diseñar expresiones complejas.

El operador de asignación (ver tabla 1.3) permite asignar el valor del operando a una determinada variable. Es un operador unario y el resultado que retorna es el mismo valor asignado.

En la programación imperativa toda variable debe tener siempre un valor antes de ser utilizada. Este valor se puede obtener de diversas maneras: a través de una lectura de datos, a través de un expresión o a través de una asignación. En muchas ocasiones se utiliza la asignación para poder inicializar las variables, sobre todo cuando estas se utilizan como acumuladores. Por ejemplo, si se quisiera diseñar un programa que sume las notas de todos los alumnos del curso de Fundamentos de Programación, podríamos acumular la suma en una variable denominada `suma`. ¿Qué valor debería tener esta variable inicialmente, antes de la acumulación? Pues debería tener el valor de 0 dado que este es el elemento neutro para la suma, entonces se hace necesario que antes de realizar la acumulación de valores, la variable tenga el valor de 0 para asegurar que el programa funcione bien.

Tabla 1.3: Operador de Asignación.

Operador	**Descripción**
=	Asignación

1.7.1. Asignación en la declaración de una variable

El lenguaje C permite que se asigne un valor inicial al momento de declarar una variable. Para realizar esto, basta que en la declaración de la variable, inmediatamente después del identificador, se añada el símbolo `=` junto con el valor que se desea inicializar a la variable. Nótese que, en el caso de la variable sexo, se le asigna el carácter `'F'` que debe estar entre comillas simples. Podrá visualizar un ejemplo de lo mencionado en el programa 1.4.

Programa 1.4: Inicialización de variables al momento de su declaración en C

```c
        int suma=0;
        double promedio=0.0;
        char sexo='F';
```

¿Cómo hacemos para verificar que la variable posee determinado valor? Existen varias maneras de hacer dicha verificación. Se puede inspeccionar el valor de las variables en tiempo de ejecución, los IDE ofrecen diversas herramientas para soportar esta tarea. Una solución simple es utilizar la función `printf` para que envíe el valor almacenado en la variable a la salida estándard. Un ejemplo de esto se puede apreciar en el programa 1.5.

Programa 1.5: Impresión de variables en C

```c
#include <stdio.h>

int main() {
    int suma=0;
    double promedio=0.0;
    char sexo='F';

    printf("%d\n", suma);
    printf("%lf\n", promedio);
```

```
10     printf(" %c\n", sexo);
11     return 0;
12 }
```

Luego de ejecutar el programa 1.5 se obtiene la siguiente salida:

```
0
0.000000
F
```

Como se mencionó anteriormente, la función `printf` es una función de la biblioteca estándar de entrada y salida del lenguaje C que permite imprimir datos según determinado formato. El formato que se imprime se encuentra deliminado por las comillas dobles (`"`). En términos prácticos se puede decir que la función `printf` envía todo lo que se encuentra en la cadena de formato a la salida estándar. Pero existen algunos caracteres que poseen un significado especial, entre estos se tiene el *back slash* (`\`) y el símbolo de porcentaje (`%`). Cuando la función `printf` se encuentra procesando la cadena de formato y encuentra estos caracteres, no imprime el carácter en cuestión sino que ejecuta la impresión dependiendo del carácter que le sigue.

En el caso del `\n`, el C lo interpreta como cambio de línea. Esto quiere decir que en la salida estándar lo siguiente que se imprima se realizará en la siguiente línea.

Para poner en práctica

¿Qué pasaría si se eliminasen los “cambios de línea” (`\n`) en el programa 1.5? Analice la salida y verifique si consigue identificar los valores de las variables.

En el caso del símbolo `%`, este da inicio a una marca de formato. Se usa generalmente para enviar los valores de las variables, en el caso de la función `printf`, a la salida estándar. Esta marca de formato tiene muchas opciones pero en esta guía centraremos la atención en solo una parte de ella, el tipo de dato. Existen símbolos que indicarán el tipo de dato de la variable que se desea imprimir. `%d` se utiliza para indicar que se imprimirá un número entero, `%lf` se utiliza para indicar que se imprimirá un número real de doble precisión y `%c` se utilizará para indicar se que imprimirá el carácter que representa el código ASCII (American Standard Code for Information Interchange (ASCII))[5].

Si se desease imprimir el valor entero que contiene la variable `suma` se deberá utilizar como formato `%d`. De esta forma la instrucción completa que imprime el valor de la variable `suma` como entero y además realiza un cambio de línea es la siguiente: `printf ("%d\n", suma);`

En el caso de la variable `sexo` del programa 1.5, en la asignación que se realiza en la línea 6 (`char sexo='F';`) en realidad el valor que se asigna a la variable `sexo` es 70, que corresponde con el valor de la letra `F` en la tabla de códigos ASCII. Al ejecutar la instrucción `printf ("%c\n", sexo);` en realidad lo que se está solicitando es que se imprima el carácter cuyo código ASCII está siendo representado en la variable `sexo`. Dicho carácter debe ir entre comillas simples.

Para poner en práctica

¿Qué pasaría si en el programa 1.5 en la línea 10, en lugar de utilizar el formato `%c` se utiliza el formato `%d`?

- ¿Ocurre algún error al compilar el programa?
- Si no ocurre ningún error, ¿qué se imprime?

Los valores que se asignen a las variables deben estar de acorde a sus tipos de datos. Es decir, si la variable es de tipo entera deberá asignársele un valor entero, si la variable es de tipo real, deberá de asignársele un valor real.

[5] En la URL https://www.ascii-code.com/ podrá visualizar la tabla de códigos ASCII extendida.

Para poner en práctica

¿Qué pasaría si en el programa 1.5 en en línea 4, en lugar de asignarle el valor de 0 a la variable `suma`, se le asigna el valor de 15.4?

- ¿Ocurre algún error al compilar el programa?
- ¿Ocurre alguna advertencia al compilar el programa?
- Si no ocurre ningún error, ¿qué se imprime?

En general, ¿cómo se comporta el C cuando se asignan valores reales a variables de tipo enteras? Sugerencia para encontrar la respuesta: asigne varios números reales en la línea 4 del programa 1.5 y encuentre el patrón de comportamiento.

1.7.2. Asignación luego de la declaración de una variable

Las variables se pueden actualizar en cualquier parte del programa, pero luego de la actualización, no se podrá recuperar el valor anterior. Para asignar valores a las variables luego de su declaración se utiliza el operador de asignación (`=`) de forma similar que en el caso anterior. La única diferencia es que no se tendría que colocar el tipo de dato (`int`, `double`, `char`) pues este solamente se usa para la declaración de una variable. Una vez que la variable se declara, esta puede ser usada a lo largo del bloque de instrucciones en donde esta se define. En el programa 1.6 se puede ver un ejemplo de actualización de la variable `promedio` en la línea 6. La declaración ocurre en la línea 4, en donde sí es obligatorio colocar el tipo de dato, en este caso `double`.

Programa 1.6: Asignación de valores en C

```
1 #include <stdio.h>
2 
3 int main() {
4     double promedio=0;
5 
6     promedio = 18.5;
7     printf("%lf\n", promedio);
8     return 0;
9 }
```

1.7.3. Asignación múltiple

En C es posible realizar la asignación de un mismo valor a múltiples variables a través de una única expresión. Esto se realiza a través del uso del operador `=` en cascada. En el programa 1.7 en la línea 6, se puede observar un ejemplo de asignación múltiple. En este caso, en la línea 4 se ha realizado la inicialización de los valores iniciales de las variables `a`, `b` y `c`, pero en la línea 6 se realiza la asignación múltiple. Hay que tener en cuenta que el lenguaje C procesa el operador de asignación (`=`) de derecha a izquierda. Esto significa que en la expresión `c=b=a` primero se realiza la asignación `b=a`. Esta operación, como ya se vio anteriormente, asigna a la variable `b` el valor que contiene la variable `a` retornando además el valor asignado en la operación, el cual es `a`. Posteriormente este valor retornado (`a`) se asigna a la variable `c`. En la práctica el valor de la variable `a` se asigna de forma simultánea tanto a la variable `b` como `c`.

Programa 1.7: Asignación múltiple de valores en C

```
1 #include <stdio.h>
2 
3 int main() {
4     int a=10, b=20, c=30;
5 
6     c=b=a;
7     printf("a tiene %d, b tiene %d y c tiene %d\n", a, b, c);
8     return 0;
9 }
```

En el programa 1.7, además se ha utilizado una única instrucción `printf` para realizar la impresión de la salida. En este caso, los variables que serán reemplazadas en la cadena de formato, se colocan una a continuación de otra. De esta forma el primer `%d` será reemplazado por el valor que almacena la variable `a`, el segundo `%d` será reemplazado por el valor que almacena la variable `b` y el tercer `%d` será reemplazado por el valor que almacena la variable `c`. En la función `printf` es importante que el número de variables luego de la cadena de formato sea igual al número de marcas (`%`) que se encuentran en la cadena de formato.

Para poner en práctica

¿Qué pasaría si en el programa 1.7 se cambia la línea 6 por las siguientes instrucciones?

- `c=b=a+1;`
- `c=b+1=a;`
- `c+1=b=a;`

Para cada caso, responda a las siguientes preguntas:

- ¿Ocurre algún error al compilar el programa?
- Si no ocurre ningún error, ¿qué se imprime?

¿Qué conclusión obtiene sobre la aplicación de la asignación múltiple?

1.8. Operadores aritméticos

Los operadores aritméticos en C implementan las operaciones básicas de la aritmética: suma, resta, multiplicación y división. Toma como operandos valores numéricos y retorna el valor resultante de la operación. Los símbolos usados por los operadores aritméticos se pueden apreciar en la tabla 1.4.

Tabla 1.4: Operadores aritméticos.

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División real – Cociente de la división entera
%	Módulo o resto de la división entera

En el lenguaje C se utiliza la notación infija para los operadores aritméticos, es decir que si quisieramos sumar los números a y b, la expresión que se debería utilizar para tal operación es `a+b`. En el programa 1.8 puede apreciar un ejemplo de suma de números enteros en C.

Programa 1.8: Suma de enteros en C

```c
#include <stdio.h>

int main() {
    int entero;

    entero = 15 + 15;
    printf("entero = %d\n", entero);
    return 0;
}
```

Para poner en práctica

¿Las variables de tipo `char` se pueden sumar?

- Altere la línea 6 del programa 1.8 y pruebe con la siguiente instrucción: `entero = 'A' + 1;`
- Luego de la alteración anterior, ¿existe algún error en el programa?, ¿se imprimió algún valor?
- ¿Qué pasaría si además de la alteración mencionada anteriormente, en la línea 7 en lugar de `%d` se utiliza `%c`? Pruebe sumando otros valores y encuentre el patrón de comportamiento del lenguaje C.
- ¿Qué pasa si cambia la línea 4 por `char entero;`?, ¿existe algún error en el programa?, ¿se imprimió algún valor?

Hay que poner especial cuidado a los límites de los tipos de datos. Como ya se mencionó anteriormente, los tipos de datos están limitados por la cantidad de bytes que se usa para su almacenamiento. En el caso del `int`, cuando se utiliza compiladores de 16 bits el valor máximo que una variable entera puede contener es 32 767, cuando se utiliza compiladores de 32 bits el valor máximo que una variable entera puede contener es 2 147 483 647. ¿Qué pasa cuando se intenta almacenar números que no se encuentran en el rango permitido del tipo de dato? Ocurre un fenómeno que se llama desbordamiento (*overflow*). En el programa 1.9, en la línea 6 se aprecia una expresión matemática cuyo resultado excede el rango de representación del tipo de datos `int`.

Programa 1.9: Suma de enteros en C

```
1 #include <stdio.h>
2
3 int main() {
4     int entero;
5
6     entero = 1500000000 + 1500000000;
7     printf("entero = %d\n", entero);
8     return 0;
9 }
```

El programa no emite error alguno, pero al realizar la impresión se obtiene lo siguiente:

```
entero = -1294967296
```

¿Qué ha pasado? Para poder entender el problema debe recordar que los números, y en general todos los datos, se almacenan en la memoria del computador usando el sistema binario.

Recordar que:

$0_{10} = 00_2$
$1_{10} = 01_2$
$2_{10} = 10_2$
$3_{10} = 11_2$
$4_{10} = 100_2$
$5_{10} = 101_2$

Vamos a suponer por un momento que la cantidad de bits para representar los números enteros es de 2. Con 2 bits se pueden representar a lo más 4 números. ¿Qué pasaría si quisiéramos asignar a una variable el valor de la suma `3+2`? En primer lugar, vale la pena mencionar que la operación se realiza a nivel de bits, es decir que en realidad lo que se suma es 11_2+10_2. El resultado de esta operación a nivel de bits es 101_2. Es decir que se requiere 3 bits. Como la representación para el entero, en este caso supuesto, solo permite 2 bits, el bit más significativo, es decir el que está más a la izquierda, se pierde quedando el resultado en 01_2 pues un bit se ha desbordado. Es decir que aparentemente `3+2` da como resultado `1`.

Hay que poner cuidado también con el operador de división (`/`). El operador `/` permite realizar tanto divisiones enteras como divisiones reales usando el mismo símbolo. Entonces, ¿cómo hace el lenguaje C para diferenciar si se debe realizar una división entera o real? Hace la diferenciación por los operandos. Si los operandos son números enteros, entonces realiza una división entera. Si al menos un operando es real, realiza una división real.

Programa 1.10: División de números enteros en C

```c
#include <stdio.h>

int main() {
    int cociente, resto;

    cociente = 5/2;
    resto = 5%2;
    printf("cociente = %d\n", cociente);
    printf("resto = %d\n", resto);
    return 0;
}
```

Luego de ejecutar el programa 1.10 se obtiene lo siguiente:

```
cociente = 2
resto = 1
```

Se analizará la expresión `cociente = 5/2`. Como podrá notar tanto `5` como `2` son números enteros por lo tanto la operación `5/2` realizará una división entera. En este caso retorna el cociente entero de dividir 5 entre 2 que en este caso es 2. El operador `%` lo que retorna es el resto de una división entera. En el caso de `5%2` retorna el valor de 1.

Note que, la división es entera debido a que los operandos son enteros. No tiene nada que ver que la variable `cociente` se haya declarado como `int`. En el programa 1.11 usted podrá visualizar una versión del programa en donde la variable `cociente` se ha definido como `double` pero esto no hace que la división sea real.

Programa 1.11: División de números reales en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double cociente;
5 
6     cociente = 5/2;
7     printf("cociente = %lf\n", cociente);
8     return 0;
9 }
```

Para poner en práctica

- ¿Qué se obtiene en la salida del programa 1.11?
- ¿Qué diferencia existen en la salida del programa 1.10 con salida del programa 1.11?
- ¿Qué pasaría si cambiásemos la línea 6 del programa 1.11 por `cociente = 5.0/2;`?

¿Cómo se hace para indicarle al programa en C que se desea hacer una división real a pesar que los operandos son enteros? Lo que hay que realizar en ese caso es una conversión explícita de tipo dato, en inglés esta operación se conoce como *casting* o *type cast*. En general, para realizar una conversión explícita de una expresión se coloca entre paréntesis el tipo de datos hacia el cual se desea convertir determinada expresión. Los paréntesis se colocan inmediatamente antes de la expresión.

```
(tipo de dato) expresión
```

En el programa 1.12 se puede aprecir un ejemplo de conversión explícita de datos en la línea 6.

Programa 1.12: Conversión de tipo en división de números reales en C

```c
1 #include <stdio.h>
2
3 int main() {
4     double cociente;
5
6     cociente = (double)5/2;
7     printf("cociente = %lf\n", cociente);
8     return 0;
9 }
```

Para poner en práctica

- ¿Qué se obtiene en la salida del programa 1.12?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.12 por `cociente=5/(double)2;`?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.12 por `cociente=1.0*5/2;`?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.12 por `cociente=5/2*1.0;`?

Los operadores aritméticos se ejecutan de la siguiente manera:

- En una misma expresión tienen primera prioridad los operadores `*`, `/` y `%`.
- En una misma expresión tienen segunda prioridad los operadores `+` y `-`. Esto quiere decir que en una misma expresión se ejecutará primero `*`, `/` y `%` y luego `+` y `-`.
- La operaciones se ejecutan de izquierda a derecha.

Para poner en práctica

¿Qué resultado se obtiene luego de ejecutar el programa 1.13?

Programa 1.13: Precedencia de los operadores aritméticos en C

```c
#include <stdio.h>

int main() {
    int a=2, b=4, c=16, d;

    d=c/b/a;
    printf("%d\n", d);
    return 0;
}
```

1.9. Operadores relacionales

Los operadores relacionales se utilizan para comparar dos valores. En C retornarán el valor de 1 si la comparación es correcta o 0 en caso contrario. Por ejemplo la expresión `5<10` es *verdadera* por lo que la operación retorna el valor de 1. La expresión `5<=1` es *falsa* por lo que la operación retorna el valor de 0.

A nivel de precedencia, los operadores aritméticos poseen mayor precedencia que los operadores relacionales. Esto quiere decir que en una expresión que contenga operadores aritméticos y operadores relacionales, primero se resolverán las operaciones aritméticas y luego las relacionales. La expresiones relacionales se evalúan de izquierda a derecha. En la tabla 1.5 se pueden apreciar los operadores relaciones en C.

Un error recurrente al momento del aprendizaje del lenguaje de programación C es confundir el operador de asignación (`=`) con el operador de comparación de igualdad (`==`). Ambos son operadores y retornan valores, por lo tanto pueden ser usados en diversas expresiones. El lenguaje C no percibe error en una operación que debe ser de asignación y se realiza comparación y viceversa.

Tabla 1.5: Operadores relacionales.

Operador	Descripción
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
==	Igual
!=	Diferente

Programa 1.14: Comparación de valores en C

```
1 #include <stdio.h>
2
3 int main() {
4     int resultado;
5
6     resultado = 15 == 15;
7     printf("resultado = %d\n", resultado);
8     return 0;
9 }
```

Para poner en práctica

- Ejecute el programa 1.14 y verifique qué valor se imprime en la salida estándar.
- Cambie la línea 6 por la siguiente expresión `resultado = 15=15;` y verifique si es que existe error. Si es que no hay error verifique el valor que se imprime en la línea 7.

1.10. Operadores lógicos

Los operadores lógicos se utilizan para elaborar expresiones lógicas. Estas expresiones pueden contener negaciones (*not*), conjunciones (*and*) y disyunciones (*or*).

En C una expresión lógica asume el valor de *verdadero* cuando esta tiene el valor diferente de 0 (típicamente 1) y asume el valor de *falso* cuando tiene el valor de 0. Por ejemplo la expresión `5` se asume como *verdadera*, la expresión `1` se asume como *verdadera* y la expresión `0` se asume como *falsa*.

A nivel de precedencia, los operadores lógicos poseen menor precedencia que los operadores relacionales, a excepción de la negación que tiene mayor prioridad. La expresiones relacionales se evaluan de izquierda a derecha. En la tabla 1.6 se pueden apreciar los operadores lógicos en C.

Tabla 1.6: Operadores lógicos.

<table>
<tr><th>Operador</th><th>Descripción</th></tr>
<tr><td>&&</td><td>Conjunción</td></tr>
<tr><td>||</td><td>Disyunción</td></tr>
<tr><td>!</td><td>Negación</td></tr>
</table>

En el programa 1.15 se puede apreciar un programa que realiza una evaluación de una conjunción para las proposiciones `p` y `q`. Como ya se mencionó anteriormente el valor de 1 se asume como valor lógico *verdadero* y el valor de 0 se asume como valor lógico *falso*.

Programa 1.15: Conjunción en C

```
#include <stdio.h>

int main() {
    int p, q, resultado;
```

```c

    p=1;
    q=0;
    resultado = p && q;
    printf("resultado = %d\n", resultado);
    return 0;
}
```

Para poner en práctica

- Altere los valores de `p` y `q` en el programa 1.15 y verifique la salida. Haga que tanto `p` como `q` posean distintas combinaciones de valores, algunas incluyendo el valor de 0. ¿Cuándo se imprime 0?, ¿cuándo se imprime 1?
- Altere el programa 1.15 para probar la disyunción. Use el operador `||`.
- Tomando como base el programa 1.15 implemente la condicional $(p \rightarrow q)$.

  Recuerde que $(p \rightarrow q \equiv \neg p \vee q)$
- Tomando como base el programa 1.15 implemente la bicondicional $(p \leftrightarrow q)$.

  Recuerde que $(p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p))$

Los diferentes tipos de operadores se pueden combinar para diseñar expresiones complejas (vea por ejemplo el programa 1.16). Es posible y común, por ejemplo, combinar operadores relacionales con operadores lógicos. Para poder escribir expresiones complejas hay que tener en consideración dos cosas. En primer lugar la asociatividad, es decir, desde donde se ejecuta la expresión. Los operadores aritméticos, relacionales y lógicos se asocian de izquierda a derecha. El operador de asignación se asocia de derecha a izquierda. En segundo lugar hay que tener en cuenta la prioridad de los operadores. No todos los operadores tienen la misma prioridad, algunos tendrán más prioridad que otros. Esto significa que si en una expresión se encuentra operadores de diferentes prioridades, el lenguaje C ejecutará primero a los que se encuentren en mayor prioridad.

A continuación se listan los operadores ordenados por prioridad.

1. Negación.
2. Multiplicación, División, Módulo
3. Suma, Resta
4. Menor que, Menor o igual que, Mayor que, Mayor o igual que
5. Igual, Diferente
6. Conjunción
7. Disyunción
8. Asignación

Programa 1.16: Expresiones complejas en C

```c
#include <stdio.h>

int main() {
    int a, b, n, resultado;

    a=15;
    b=20;
    n=101;
    resultado = a<b && n<=100;
    printf("resultado = %d\n", resultado);
    return 0;
}
```

Capítulo 2

Ejercicios propuestos

2.1. Nivel Básico

2.1.1. Precedencia de operadores aritméticos

Para cada par de expresiones que se presentan a continuación, se solicita que elabore un programa en C que verifique el valor resultante de cada una de ellas.

- `5+3*2+7` vs `(5+3)*(2+7)`
- `4+6/2+3` vs `(4+6)/(2+3)`

Para poner en práctica

Una vez finalizado el programa, reflexione y responda a las siguientes interrogantes:

- ¿Cuál es la regla de precedencia de los operadores aritméticos?
- ¿Quién tiene mayor precedencia, la suma o la multiplicación?
- ¿Quién tiene mayor precedencia, la suma o la división?

2.1.2. Expresiones lógicas

Para cada una de las siguientes expresiones lógicas, se le solicita que elabore un programa en C que defina las correspondientes variables para cada proposición, así como la expresión equivalente a la propuesta considerando que C solamente implementa de forma nativa la negación (`!`), conjunción (`&&`) y disyunción (`||`).

- $\neg p$
- $\neg q$
- $p \wedge q$
- $p \vee q$
- $\neg p \wedge \neg q$
- $\neg p \vee \neg q$
- $\neg p \vee q$

2.1.3. Radiación ionizante

La radiación ionizante se mide usando el sievert (Sv) que cuantifica la cantidad de radiación absorbida por los tejidos humanos[1]. Generalmente, las personas están expuestas a alrededor de entre 1 y 10 milisieverts (mSv) de radiación al año por exposición natural causada por sustancias radiactivas presentes en el aire y el suelo. Cuando las personas presentan diferentes niveles de exposición a la radiación se generan probables efectos en los seres humanos. Una exposición a 500 mSv causa náuseas en cuestión de horas.

El programa 2.1 ha sido diseñado con el objetivo de verificar si una dosis de radiación ionizante expresada en mSv, genera o no náuseas. Complete la línea 5 para que se obtenga 1 como salida en caso la dosis genere náuseas y retorne 0 en caso contrario.

Programa 2.1: Verificación si la dosis genera náuseas en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double dosis_mSv=467;
5     int genera_nauseas = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿La dosis genera náuseas? %d\n", genera_nauseas);
7     return 0;
8 }
```

2.1.4. ¿La disolución es ácida?

El potencial de Hidrógeno, más conocido como pH, es una medida que permite determinar el estado de acidez o alcalinidad de una disolución. Cuando el pH tiene el valor de 7 se dice que la disolución es neutra, cuando es menor que 7 se dice que la disolución es ácida y cuando es mayor que 7 se dice que la disolución es alcalina. El programa 2.2 ha sido diseñado con el objetivo de verificar si el pH de una disolución corresponde a una medida ácida o no. Complete la línea 5 para que se obtenga 1 como salida en caso la disolución sea ácida y retorne 0 en caso contrario.

Programa 2.2: Verificación de una disolución ácida en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double pH=5.2;
5     int es_acida = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿La disolución es ácida? %d\n", es_acida);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.2 y obtener la salida correcta, pruebe cambiando el valor de la variable *pH* y verifique la salida obtenida.
- Realice las alteraciones necesarias para determinar si la disolución es neutra.
- Realice las alteraciones necesarias para determinar si la disolución es alcalina.

[1] La información para esta pregunta ha sido obtenida a través de las URLs:
https://lta.reuters.com/articulo/internacional-sismo-japon-radiacion-idLTASIE72E0GZ20110315
https://lta.reuters.com/articulo/internacional-sismo-japon-radiacion-idLTASIE72F0TF20110316

2.1.5. Índice de Masa Corporal

El Índice de Masa Corporal (IMC) es una razón matemática que asocia la masa y la talla de un individuo. Se calcula de la siguiente manera:

$$IMC = \frac{masa}{estatura^2}$$

Donde la *masa* se expresa en kg y la *estatura* en m. El programa 2.3 ha sido diseñado con el objetivo de calcular e imprimir el IMC. Complete la línea 5 con la expresión aritmética que permita calcular el valor solicitado. En esta pregunta no podrá usar la función `pow`.

Programa 2.3: Cálculo del IMC en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double masa=89.4, estatura=1.67;
5     double imc = /*retirar comentario e incluir expresión solicitada*/;
6     printf("El IMC es %lf\n", imc);
7     return 0;
8 }
```

Para poner en práctica

Luego de completar el programa 2.3 y obtener la salida correcta (el valor obtenido debió ser de $\approx$ 32.05564918), pruebe con los datos de prueba que se presentan a continuación:

- Si $masa = 86$ kg y $estatura = 1.70$ m, se deberá imprimir $\approx$ 29.75778547.
- Si $masa = 71$ kg y $estatura = 1.70$ m, se deberá imprimir $\approx$ 24.56747405.
- Si $masa = 75$ kg y $estatura = 1.65$ m, se deberá imprimir $\approx$ 27.54820937.
- Si $masa = 79$ kg y $estatura = 1.80$ m, se deberá imprimir $\approx$ 24.38271605.

2.1.6. Área del cuadrado

El programa 2.4 ha sido diseñado con el objetivo de calcular e imprimir el área del cuadrado de lado l. Complete la línea 5 con la expresión aritmética que permita calcular el área solicitada.

Programa 2.4: Área del cuadrado en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double l=3.6;
5     double area = /*retirar comentario e incluir expresión solicitada*/;
6     printf("El área del cuadrado es %lf\n", area);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.4 y obtener la salida correcta, pruebe cambiando el valor de la variable l y verifique la salida obtenida.
- Realice las alteraciones necesarias para determinar el área de un rectángulo. Deberá declarar en este casos dos variables una que se llame $base$ y otra que se llame $altura$. Recuerde que el área del rectángulo es igual a $base \times altura$.
- Realice las alteraciones necesarias para determinar el área de un triángulo dado su base y su altura. Recuerde que el área del triángulo es igual a $\dfrac{base \times altura}{2}$.

2.1.7. Cálculo de π

Se sabe que el perímetro de una circunferencia es $2\pi r$, donde r es el radio de dicha circunferencia. Si el perímetro de una circunferencia de radio 5.6 es 35.185838, ¿cuál es el valor de π que se ha usado en este cálculo? El programa 2.5 ha sido diseñado con el objetivo de calcular e imprimir el valor de π dado el perímetro p de un circunferencia de radio r. Complete la línea 5 con la expresión aritmética que permita calcular el valor de π.

Programa 2.5: Cálculo de π en C

```
1 #include <stdio.h>
2
3 int main() {
4     double p=35.185838, r=5.6;
5     double pi = /*retirar comentario e incluir expresión solicitada*/;
6     printf("El valor de PI usado es %lf\n", pi);
7     return 0;
8 }
```

2.1.8. Conversión de grados sexagesimal a radianes

El programa 2.6 ha sido diseñado con el objetivo de convertir grados sexagesimales en radianes. Complete la línea 5 para que se realice la conversión correspondiente.

Recordar que:

$360° = 2\pi$ radianes

Programa 2.6: Conversión de grados sexagesimales a radianes en C

```
1 #include <stdio.h>
2
3 int main() {
4     double sexagesimal=45;
5     double radianes = /*retirar comentario e incluir expresión solicitada*/;
6     printf("%lf grados sexagesimnales equivale a %lf radianes\n", radianes);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.6 y obtener la salida correcta (el valor obtenido debió ser de ≈ 0.785398), pruebe con los datos de prueba que se presentan a continuación:
    - Si $sexagesimal = 15.7$ se deberá imprimir ≈ 0.2740167.
    - Si $sexagesimal = 89.6$ se deberá imprimir ≈ 1.563815.
    - Si $sexagesimal = 180$ se deberá imprimir ≈ 3.14159.
- Realice un programa para realizar la conversión de radianes a sexagesimal.

2.1.9. Tiempo de encuentro

Se tienen dos objetos separados a una distancia d. Cada uno de ellos se mueve a una velocidad constante. ¿Qué tiempo debe pasar para que ambos se encuentren? El programa 2.7 ha sido diseñado con el objetivo de calcular el tiempo de encuentro de dos objetos que están separados 400 km entre sí. El primer objeto parte con una velocidad v_1 de 50 kph mientras que el segundo parte al mismo tiempo con una velocidad v_2 de 30 kph. Complete la línea 5 para que calcule el tiempo de encuentro t_e.

Recordar que:

$$t_e = \frac{d}{v_a + v_b}$$

Programa 2.7: Tiempo de encuentro en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double d=400, v1=50, v2=30;
5     double te = /*retirar comentario e incluir expresión solicitada*/;
6     printf("El tiempo de encuentro es %lf\n", te);
7     return 0;
8 }
```

Para poner en práctica

Luego de completar el programa 2.7 y obtener la salida correcta (el valor obtenido debió ser de 5), pruebe con los datos de prueba que se presentan a continuación:

- Si $d = 300$, $v1 = 60$ y $v2 = 45$, se deberá imprimir ≈ 2.857142857.
- Si $d = 1000$, $v1 = 15$ y $v2 = 20$, se deberá imprimir ≈ 28.57142857.
- Si $d = 5000$, $v1 = 90$ y $v2 = 120$, se deberá imprimir ≈ 23.80952381.

2.1.10. Aplicación de la ley de Charles

El volumen inicial de una cierta cantidad de gas es de 200 cm^3 a la temperatura de 20°C. Calcule el volumen a 90°C si la presión permanece constante. El programa 2.8 ha sido diseñado con el objetivo de resolver este problema. Complete la línea 5 para que se realice el cálculo deseado.

Recordar que:

Ley de Charles establece que el volumen es directamente proporcional a la temperatura del gas, cuando la presión permanece constante. Si la temperatura aumenta, el volumen también aumenta. Si la temperatura disminuye, el volumen también disminuye.

De esta ley se obtiene la siguiente relación: $\dfrac{V_1}{T_1} = \dfrac{V_2}{T_2}$

Programa 2.8: Aplicación de la ley de Charles en C

```c
#include <stdio.h>

int main() {
    double v1cm3=200, t1C=20, t2C=90;
    double v2cm3 = /*retirar comentario e incluir expresión solicitada*/;
    printf("El volumen final es %lf cm^3\n", v2cm3);
    return 0;
}
```

Para poner en práctica

Luego de completar el programa 2.8 y obtener la salida correcta (el valor obtenido debió ser de 900), pruebe con los datos de prueba que se presentan a continuación:

- Si $v1 = 250$, $t1 = 40$ y $t2 = 85$, se deberá imprimir ≈ 531.25.
- Si $v1 = 300$, $t1 = 35$ y $t2 = 50$, se deberá imprimir ≈ 428.5714286.
- Si $v1 = 450$, $t1 = 85$ y $t2 = 60$, se deberá imprimir ≈ 317.6470588.

2.2. Nivel Intermedio

2.2.1. ¿Comparación múltiple? parte 1

El lenguaje C es un lenguaje muy flexible, pero es manejado por una serie de reglas y regido por la precedencia de operadores. El operador de comparación (==) en un operador binario, por lo que siempre operará con 2 operandos según la precedencia. Ejecute el programa 2.9 y luego responda las siguientes interrogantes.

Programa 2.9: Comparación anidada en C

```c
#include <stdio.h>

int main() {
    int a=5, b=5, c=5;
    int todos_iguales = a==b==c;
    printf("¿Son los números iguales? %d\n", todos_iguales);
    return 0;
}
```

Para poner en práctica

- ¿Qué se obtiene como salida? ¿El lenguaje C puede realizar una comparación múltiple?
- Reemplace los valores de la siguiente manera: `a=5`, `b=3` y `c=0`.
- ¿Qué se obtiene como salida? Describa el orden de ejecución de las operaciones.
- Reemplace los valores de la siguiente manera: `a=1`, `b=1` y `c=1`.
- ¿Qué se obtiene como salida? Explique el comportamiento de este programa poniendo énfasis en la línea 5.

2.2.2. ¿Comparación múltiple? parte 2

Tal como se puede apreciar en el problema propuesto 2.2.1, el lenguaje C es un lenguaje muy flexible pero a veces las operaciones que ejecuta no son las que los humanos podrían pensar. Tal como se indicó a lo largo de esta guía, los operadores relacionales son operadores binarios. Esto significa que operará solamente con 2 operandos según la precedencia. Entonces, ¿por qué el programa 2.10 aparenta ejecutar una comparación múltiple? Ejecútelo y luego responda las interrogantes presentadas.

Programa 2.10: Comparación anidada en C

```
1 #include <stdio.h>
2
3 int main() {
4     int a=5, b=5, c=3;
5     int verifica_relacion = a<b<c;
6     printf("¿a<b<c? %d\n", verifica_relacion);
7     return 0;
8 }
```

Para poner en práctica

- ¿Qué se obtiene como salida? ¿El lenguaje C puede realizar una comparación múltiple?
- Reemplace los valores de la siguiente manera: `a=5`, `b=15` y `c=3`.
- ¿Qué se obtiene como salida? Describa el orden de ejecución de las operaciones.
- Reemplace los valores de la siguiente manera: `a=1`, `b=3` y `c=5`.
- ¿Qué se obtiene como salida? Explique el comportamiento de este programa poniendo énfasis en la línea 5.

2.2.3. Expresiones lógicas

Para cada una de las siguientes expresiones lógicas, se le solicita que elabore un programa en C que defina las correspondientes variables para cada proposición así como la expresión equivalente a la propuesta considerando que C solamente implementa de forma nativa la negación (`!`), conjunción (`&&`) y disyunción (`||`). En este ejercicio, no podrá reducir la expresión lógica.

- $p \rightarrow (r \lor \neg\, q)$
- $\neg\, (p \lor q) \leftrightarrow (\neg\, p \land \neg\, q)$
- $(\neg\, p \land (q \lor r)) \leftrightarrow ((p \lor r) \land q)$
- $\neg\, (\neg\, (p \lor (\neg\, q \rightarrow p)) \lor \neg\, ((p \leftrightarrow \neg\, q) \rightarrow (q \land \neg\, p)))$

Recordar que:

- $p \rightarrow q \equiv \neg p \vee q$
- $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

2.2.4. Verificación de triángulo isósceles

El programa 2.11 ha sido diseñado con el objetivo de verificar si es que los lados $l1$, $l2$ y $l3$ pueden formar un triángulo isósceles. Complete la línea 5 para que se obtenga 1 como salida en caso los lados formen un triángulo isósceles y retorne 0 en caso contrario.

Programa 2.11: Verificación de triángulo isósceles en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double l1=12.5, l2=13, l3=12.5;
5     int es_isosceles = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿Es isósceles? %d\n", es_isosceles);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.11 y obtener la salida correcta, pruebe cambiando los valores de los lados y verifique la salida obtenida.
- Actualice la línea 5 para que contenga la condición de verificación del triángulo equilátero.
- Actualice la línea 5 para que contenga la condición de verificación del triángulo escaleno.
- ¿Cree usted que los lados ingresados formen un triángulo? Actualice la línea 5 para que la condición verifique si los 3 lados forman un triángulo.

Recordar que:

Según el teorema de la desigualdad del triángulo *"La suma de las longitudes de cualesquiera de los lados de un triángulo es mayor que la longitud del tercer lado"*.

2.2.5. Verificación de cuadrado

El programa 2.12 ha sido diseñado con el objetivo de verificar si es que los lados $l1$, $l2$, $l3$ y $l4$ pueden formar un cuadrado. Complete la línea 5 para que se obtenga 1 como salida en caso los lados formen un cuadrado y retorne 0 en caso contrario.

Programa 2.12: Verificación de cuadrado en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     double l1=12.5, l2=12.5, l3=12.5, l4=12.5;
5     int es_cuadrado = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿Es cuadrado? %d\n", es_cuadrado);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.12 y obtener la salida correcta, pruebe cambiando los valores de los lados y verifique la salida obtenida.
- Actualice la línea 5 para que contenga la condición de verificación de un rectángulo. No asuma que los lados están en orden. Es decir, un par de lados iguales podría ser $l1$ y $l2$, $l1$ y $l3$ o $l1$ y $l4$.

2.2.6. Verificación de pertenencia a un rango

En diversas estructuras de control de flujo, es muy común necesitar conocer, si determinado número n se encuentra contenido en, determinado rango $[a, b]$. El programa 2.13 ha sido diseñado con el objetivo de verificar si es que el número n se encuentra contenido en el rango $[a, b]$. Complete la línea 5 para que se obtenga 1 como salida en caso n se encuentre en el rango y retorne 0 en caso contrario.

Programa 2.13: Verificación de pertenencia a un rango en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     int n=7, a=3, b=10;
5     int pertenece_a_rango = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿ %d pertenece al rango [%d, %d]? %d\n", n, a, b, pertenece_a_rango);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.13 y obtener la salida correcta, pruebe cambiando los valores del rango y verifique la salida obtenida.
- Actualice la línea 5 para que la verificación se realice en el rango $]a, b]$.
- Actualice la línea 5 para que la verificación se realice en el rango $[a, b[$.
- Actualice la línea 5 para que la verificación se realice en el rango $]a, b[$.

2.2.7. Mayor de 3 números

Dadas las variables a, b y c que representan a 3 números enteros. Se desea determinar si la variable a contiene el mayor valor de las 3 variables. El programa 2.14 ha sido diseñado con el objetivo de comparar las 3 variables y determinar si a es la mayor de todas. Complete la línea 5 para que se obtenga 1 como salida en caso a contenga el mayor valor y retorne 0 en caso contrario.

Programa 2.14: Mayor de 3 números en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     int a=5, b=3, c=1;
5     int es_a_mayor_de_todos = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿Es %d el número mayor? %d\n", a, es_a_mayor_de_todos);
7     return 0;
8 }
```

Para poner en práctica

- Luego de completar el programa 2.14 y obtener la salida correcta, pruebe cambiando el valor de las variables a, b y c y verifique la salida obtenida.
- Realice las adecuaciones al programa 2.14 para determinar si la variable a contiene el menor valor de las 3 variables.
- Realice las adecuaciones al programa 2.14 para determinar si la variable a contiene el mayor valor de las 3 variables y los demás números sean diferentes entre sí.

2.2.8. Promedio y porcentaje

Dado 3 números enteros se solicita calcular el promedio aritmético de los 3 números así como el porcentaje que representa la magnitud del primer número en relación a los demás. El programa 2.15 ha sido diseñado con el objetivo de resolver este problema. Complete la línea 5 para obtener el promedio aritmético y complete la línea 6 para obtener el porcentaje.

Programa 2.15: Promedio y porcentaje de números enteros en C

```c
1  #include <stdio.h>
2  
3  int main() {
4      int a=15, b=19, c=16;
5      double promedio = /*retirar comentario e incluir expresión solicitada*/;
6      double porcA = /*retirar comentario e incluir expresión solicitada*/;
7      printf("El promedio de los 3 números es %lf\n", promedio);
8      printf("%d corresponde al %lf%%\n", a, porcA);
9      return 0;
10 }
```

Para poner en práctica

- Luego de completar el programa 2.15 y obtener la salida correcta (se debe obtener como promedio ≈ 16.66666667 y como porcentaje 30 %), pruebe cambiando los valores de los números enteros y verifique la salida obtenida.
- Actualice la línea 8 retirando los dos últimos `%%`. ¿Qué se imprime?, ¿para qué sirve el `%%`?

2.2.9. ¿Cuánto tarda la luz del sol en llegar a la tierra?

Se desea elaborar un programa en C que determine el tiempo que demora en viajar la luz desde el sol hasta la superficie de la tierra. La salida de este programa debe ser similar a:

```
La luz demora en llegar desde el sol 500.346143 segundos
La luz demora en llegar desde el sol 8.339102 minutos
```

Recordar que:

- La $v = \frac{d}{t}$, donde $v = velocidad$, $d = distancia$ y $t = tiempo$.
- La distancia del sol a la tierra es aproximadamente de 150 millones de km.
- La velocidad de la luz es aproximadamente 299 792 458 m/s.
- 1 $km = 1000m$.

2.2.10. Comparación de velocidades

Dadas la velocidades v_1 y v_2, se desea determinar cuál de las dos es mayor en magnitud sabiendo que la v_1 está expresada en kph y la v_2 está expresada en mph. El programa 2.16 ha sido diseñado con el objetivo de comparar las dos velocidades pero usando la misma unidad de medida. Complete la línea 5 para que se obtenga 1 como salida en caso la v_1 sea mayor que la v_2 y retorne 0 en caso contrario.

Recordar que:

$$1\ mi = 1.609344 \times km$$

Programa 2.16: Comparación de velocidades en C

```c
1 #include <stdio.h>
2
3 int main() {
4     double v1_kph = 140, v2_mph=90;
5     int v1_es_mayor = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿La velocidad v1 es mayor? %d\n", v1_es_mayor);
7     return 0;
8 }
```

Para poner en práctica

Realice la conversión de las velocidades:

- Convierta la v_2 a kph y compare.
- Convierta la v_1 a mph y compare.

Utilice los siguientes datos para probar su solución.

- 140 kmh equivale a $\approx$ 86.992 mph.
- 150 kmh equivale a $\approx$ 93.2057 mph.
- 160 kmh equivale a $\approx$ 99.4194 mph.

2.2.11. Aplicación de la ley de Boyle

Una cantidad de gas ocupa un volumen de 80 cm^3 a una presión de 750 mm Hg. ¿Qué volumen ocupará a una presión de 1.2 atm. si la temperatura no cambia? El programa 2.17 ha sido diseñado con el objetivo de resolver este problema. Complete la línea 5 para que se realice el cálculo deseado.

Recordar que:

La ley de Boyle establece que la presión de un gas en un recipiente cerrado es inversamente proporcional al volumen del recipiente, cuando la temperatura permanece constante.

De esta ley se obtiene la siguiente relación: $P_1 \times V_1 = P_2 \times V_2$

Recuerde que las presiones deben estar en la misma unidad. 1 atm = 760 mm Hg

Programa 2.17: Aplicación de la ley de Boyle en C

```c
1 #include <stdio.h>
2
3 int main() {
```

```c
    double v1cm3=80, p1mmHg=750, p2atm=1.2;
    double v2cm3 = /*retirar comentario e incluir expresión solicitada*/;
    printf("El gas ocupa %lf cm^3\n", v2cm3);
    return 0;
}
```

Para poner en práctica

Utilice los siguientes datos para probar su solución.

- Si $v_1 = 80\ cm^3$, $p_1 = 750\ mmHg$ y $p_2 = 1.2\ atm$, v_2 equivale a $\approx 65.78947368\ cm^3$.
- Si $v_1 = 90\ cm^3$, $p_1 = 740\ mmHg$ y $p_2 = 1.3\ atm$, v_2 equivale a $\approx 67.40890688\ cm^3$.
- Si $v_1 = 100\ cm^3$, $p_1 = 735\ mmHg$ y $p_2 = 1.1\ atm$, v_2 equivale a $\approx 87.91866029\ cm^3$.

2.2.12. Fracciones

El programa 2.18 ha sido diseñado con el objetivo de calcular e imprimir la suma de dos fracciones. Complete la línea 5 con la expresión aritmética que permita calcular el numerador de la suma y complete la línea 6 con la expresión aritmética que permita calcular el denominador de la suma.

Programa 2.18: Suma de fracciones en C

```c
1 #include <stdio.h>
2 
3 int main() {
4     int num1=1, den1=2, num2=1, den2=3;
5     int num_suma = /*retirar comentario e incluir expresión solicitada*/;
6     int den_suma = /*retirar comentario e incluir expresión solicitada*/;
7     printf("La suma de %d/%d + %d/%d es %d/%d\n", num1, den1, num2, den2, num_suma, den_suma);
8     return 0;
9 }
```

Para poner en práctica

- Luego de completar el programa 2.18 y obtener la salida correcta, pruebe cambiando el valor de las variables y verifique la salida obtenida. Use los siguientes casos de prueba.
    - Si la fracción 1 es $\frac{1}{2}$ y la fracción 2 es $\frac{1}{3}$, se deberá imprimir $\frac{5}{6}$
    - Si la fracción 1 es $\frac{3}{2}$ y la fracción 2 es $\frac{1}{2}$, se deberá imprimir $\frac{8}{4}$
    - Si la fracción 1 es $\frac{1}{5}$ y la fracción 2 es $\frac{1}{6}$, se deberá imprimir $\frac{11}{30}$
- Modifique el programa para realizar ahora la resta de fracciones.
- Modifique el programa para realizar ahora la multiplicación de fracciones.
- Modifique el programa para realizar ahora la división de fracciones.

2.3. Nivel Avanzado

2.3.1. Cifrado por sustitución

El cifrado César es uno de los primeros métodos de cifrado conocidos históricamente. Julio César lo usó para enviar órdenes a sus generales en los campos de batalla. Consistía en escribir el mensaje con un alfabeto que estaba formado por las letras del alfabeto latino normal desplazadas tres posiciones a la derecha[2].

El programa 2.19 ha sido diseñado con el objetivo de cifrar un mensaje de 4 caracteres usando el cifrado César. El mensaje se encuentra descrito en las variables $c1$, $c2$, $c3$, y $c4$. Complete las líneas 5, 6, 7 y 8 con la expresión aritmética que permita hacer la sustitución por el método de César. Por comodidad para este ejercicio asuma que nunca recibirá los caracteres `X`, `Y` o `Z`.

Programa 2.19: Criptografía por sustitución en C

```c
1  #include <stdio.h>
2
3  int main() {
4      char c1 = 'H', c2 = 'O', c3 = 'L', c4 = 'A';
5      char c1_cifrado = /*retirar comentario e incluir expresión solicitada*/;
6      char c2_cifrado = /*retirar comentario e incluir expresión solicitada*/;
7      char c3_cifrado = /*retirar comentario e incluir expresión solicitada*/;
8      char c4_cifrado = /*retirar comentario e incluir expresión solicitada*/;
9      printf("El mensaje cifrado es %c%c%c%c\n", c1_cifrado, c2_cifrado, c3_cifrado, c4_cifrado);
10     return 0;
11 }
```

Para poner en práctica

Luego de completar el programa 2.19 y obtener la salida correcta (el valor impreso debió ser “`KROD`”), pruebe con los datos de prueba que se presentan a continuación:

- Si el mensaje a cifrar es “`casa`” se debe imprimir “`fdvd`”.
- Si el mensaje a cifrar es “`Mama`” se debe imprimir “`Pdpa`”.
- Si el mensaje a cifrar es “`LUIS`” se debe imprimir “`OXLV`”.

2.3.2. Transformación de dígitos hexadecimal a base 10

Los sistemas de numeración posicional basan la representación de los números en torno al valor del dígito y la posición que ocupa este en el número. En el sistema en base 16 o hexadecimal se puede tener hasta 16 dígitos, estos son los dígitos del `0` al `9`, más las cifras `A`, `B`, `C`, `D`, `E` y `F`. La cifra `A` corresponde al valor 10 en base decimal, la `B` corresponde al 11, la `C` al 12, la `D` al 13, la `E` al 14 y la `F` corresponde al 15.

El programa 2.20 ha sido diseñado con el objetivo de transformar el valor de un dígito en hexadecimal a decimal. Complete las líneas 5 con la expresión aritmética que permita realizar dicha transformación.

Programa 2.20: Transformación de dígitos hexadecimal en C

```c
1  #include <stdio.h>
2
3  int main() {
4      char digito_hexa = 'D';
5      int valor_en_decimal = /*retirar comentario e incluir expresión solicitada*/;
6      printf("%c en hexadecimal vale %d en decimal\n", digito_hexa, valor_en_decimal);
7      return 0;
8  }
```

Para poner en práctica

Luego de completar el programa 2.20 pruebe transformando los demás dígitos hexadecimales. El programa no deberá ser modificado, solo podrá alterar la asignación que se realiza en la línea 4.

[2] Tomado de la URL http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/cesar.html

2.3.3. Conversión de grados Centígrados en Fahrenheit

El programa 2.21 ha sido diseñado con el objetivo de convertir grados centígrados en Fahrenheit. Ejecute el programa y verifique si es que cumple con su objetivo. Sugerencia: prueba con varios valores para la variable `c` en el rango [0..100]. En caso no cumpliera con su objetivo, ¿qué cambios realizaría para que funcione correctamente? ¿son necesarios todos los paréntesis en la línea 5?

Programa 2.21: Conversión de grados Centígrados en Fahrenheit en C

```c
1 #include <stdio.h>
2
3 int main() {
4     int c=21;
5     double f = (c * 9 / 5)+(32);
6     printf("%dC equivale a %lfF", c, f);
7     return 0;
8 }
```

Para poner en práctica

- Luego de corregir el programa 2.21 y obtener la salida correcta (el valor obtenido debió ser de 69.8), pruebe con los datos de prueba que se presentan a continuación:
  - Si se tienen 36 grados centígrados, se deberá imprimir ≈ 96.8 grados Fahrenheit.
  - Si se tienen 42 grados centígrados, se deberá imprimir ≈ 107.6 grados Fahrenheit.
  - Si se tienen 56 grados centígrados, se deberá imprimir ≈ 132.8 grados Fahrenheit.
- Realice un programa para realizar la conversión de Fahrenheit a Centígrados.
- Realice un programa para realizar la conversión de Centígrados a Kelvin.
- Realice un programa para realizar la conversión de Fahrenheit a Kelvin.
- Realice un programa para realizar la conversión de Kelvin a Centígrados.
- Realice un programa para realizar la conversión de Kelvin a Fahrenheit.

2.3.4. Números pseudoaleatorios

Un número pseudoaleatorio se puede definir como un número que ha sido generado en un proceso aparentemente al azar, pero que en realidad no lo es. Existen muchas formas de generar números pseudoaleatorios, dentro de ellos destaca el algoritmo generador lineal congruencial[3]. Según este algoritmo se pueden generar números pseudoaleatorios a partir de la siguiente función recursiva:

$$X_{n+1} = (aX_n + c) \ mod \ m$$

Donde:

- m es el módulo, $m > 0$
- a es el multiplicador, $0 < a < m$
- c es el incremento, $0 \leq c < m$

El programa 2.22 ha sido diseñado con el objetivo de determinar el siguiente número aleatorio a generar. Complete las líneas 5 con la expresión aritmética que permita realizar dicha generación.

[3] https://en.wikipedia.org/wiki/Linear_congruential_generator

Programa 2.22: Generación de números pseudoaleatorios en C

```c
#include <stdio.h>

int main() {
    int m = 65536, a = 2269, c = 1, x0 = 29;
    int num_pseudoaleatorio = /*retirar comentario e incluir expresión solicitada*/;
    printf("El siguiente número pseudoaleatorio es: %d\n", num_pseudoaleatorio);
    return 0;
}
```

Para poner en práctica

Con los datos que se presentan, deberá generarse el número 266. Los computadores al momento de generar números pseudoaleatorios, cambian el valor de X_0 el cual denominan semilla. Intente cambiando los valores de la semilla del programa y vea si encuentra algún patrón de comportamiento.

2.3.5. Raíces de una ecuación de tercer grado

Dada una ecuación de tercer grado con una variable de la siguiente forma $x^3 + a_1x^2 + a_2x + a_3 = 0$, se requiere saber si todas sus raíces son reales y distintas.

Recordar que:

Dada una ecuación de tercer grado con una variable de la siguiente forma $x^3 + a_1x^2 + a_2x + a_3 = 0$, esta tendrá todas las raíces reales y distintas, si y solo si el discriminante es menor que cero.

El discriminante se puede calcular como $discriminante = D^3 + R^2$, donde:

$$D = \frac{3a_2 - a_1^2}{9} \text{ y } R = \frac{9a_1a_2 - 27a_3 - 2a_1^3}{54}$$

El programa 2.23 ha sido diseñado con el objetivo de determinar si la ecuación de tercer grado, descrita por las variables a_1, a_2 y a_3, posee raíces reales y distintas. Complete la líneas 5 con la expresión aritmética que permita realizar la verificación solicitada. Se sugiere que utilice otras variables adicionales para almacenar los valores intermedios que requiera. En este ejercicio no podrá usar la función `pow`.

Programa 2.23: Discriminantes de una ecuación de tercer grado en C

```c
1 #include <stdio.h>
2
3 int main() {
4     double a1=1, a2=-3, a3=1;
5     int tiene_raices_reales_y_diferentes = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿tiene raíces reales y diferentes? %d\n", tiene_raices_reales_y_diferentes);
7     return 0;
8 }
```

Para poner en práctica

Pruebe con los datos de prueba que se presentan a continuación:

- Si $a_1 = 1$, $a_2 = -3$ y $a_3 = 1$, entonces `todas las raíces son reales y diferentes`.
- Si $a_1 = 2$, $a_2 = -5$ y $a_3 = -6$, entonces `todas las raíces son reales y diferentes`.
- Si $a_1 = 1$, $a_2 = 1$ y $a_3 = 1$, entonces `no todas las raíces son reales y diferentes`.

2.3.6. Encuentro de vehículos

Dos vehículos v_1 y v_2 que se dirigen hacia la misma meta a una velocidad constante, se encuentran separados a una distancia d. La ruta que siguen ambos es una línea recta. Se desea saber si es que el vehículo v_1 alcanzará al vehículo v_2. El programa 2.24 ha sido diseñado con el objetivo de determinar si el vehículo v_1 puede alcanzar al vehículo v_2 en los siguientes t (30) minutos. El vehículo v_2 se encuentra a d (20) km delante del vehículo v_1 y posee una velocidad constante de v_1kmp (45) km/h mientras que el v_2 posee una velocidad constante de v_2kmp (25) km/h. Complete la línea 5 para que se obtenga 1 como salida en caso v_1 alcance a v_2 y retorne 0 en caso contrario.

Programa 2.24: Encuentro de vehículos en C

```
1 #include <stdio.h>
2
3 int main() {
4     double d=20, v1kmp=45, v2kmp=25, t=30;
5     int v1_alcanza_a_v2 = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿v1 alcanza a v2? %d\n", v1_alcanza_a_v2);
7     return 0;
8 }
```

2.3.7. ¿Se mueve o no se mueve la caja?

Sobre un plano horizontal se tiene una caja que pesa 35 N. Se sabe además que se tiene un coeficiente de rozamiento estático de $\mu_s = 0.5$. Si se le aplica una fuerza de 10 N, ¿la caja se mueve? El programa 2.25 ha sido diseñado con el objetivo de determinar si la caja de mueve o no. Complete la línea 5 para que se obtenga 1 como salida en caso la caja se mueva y retorne 0 en caso contrario.

Programa 2.25: Movimiento de una caja en C

```
1 #include <stdio.h>
2
3 int main() {
4     double p=35, mu=0.5, f=10
5     int se_mueve = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿Se mueve la caja? %d\n", se_mueve);
7     return 0;
8 }
```

Recordar que:

La fuerza de rozamiento estático máximo se calcula de la siguiente manera:

$$fs\_max = \mu_s \times N$$

Donde:

- μ_s es el coeficiente de rozamiento estático.
- N es la fuerza normal.

Sugerencia

- Calcule la fuerza de rozamiento estático máximo. Recuerde que la fuerza de rozamiento estático máximo equivale a la fuerza mínima para iniciar un movimiento.
- Si la fuerza aplicada es mayor que la fuerza de rozamiento estático máximo, la caja se moverá.
- Si la fuerza aplicada es menor o igual que la fuerza de rozamiento estático máximo, la caja no se moverá.

2.3.8. Los números Armstrong

Un número Armstrong, también llamado número narcisista, es todo aquel número que es igual a la suma de cada uno de sus dígitos elevado al número total de dígitos.

A continuación siguen algunos ejemplos de números Armstrong.

- $371 = 3^3 + 7^3 + 1^3$. Total de dígitos 3.
- $8208 = 8^4 + 2^4 + 0^4 + 8^4$. Total de dígitos 4.
- $4210818 = 4^7 + 2^7 + 1^7 + 0^7 + 8^7 + 1^7 + 8^7$. Total de dígitos 7.

El programa 2.26 ha sido diseñado con el objetivo de verificar si es que un determinado número de 3 cifras es un número Armstrong o no. La variable u corresponde al dígito de las unidades, la variable d corresponde al dígito de las decenas y la variable c corresponde al dígito de las centenas del número correspondiente. Complete la línea 5 para que se obtenga 1 como salida en caso el número en cuestión sea Armstrong y retorne 0 en caso contrario.

Programa 2.26: Verificación de número Armstrong en C

```
1 #include <stdio.h>
2
3 int main() {
4     int u=1, d=5, c=3;
5     int es_Armstrong = /*retirar comentario e incluir expresión solicitada*/;
6     printf("¿Es Armstrong? %d\n", es_Armstrong);
7     return 0;
8 }
```

2.3.9. Días transcurridos entre dos fechas

El programa 2.27 ha sido diseñado con el objetivo de calcular la diferencia en días existentes entre dos fechas pertenecientes al mismo año. Para representar la fecha 1 se tienen la variable `dd1` que corresponde al número del día de la fecha 1 y `mm1` que corresponde al número del mes de la fecha 1. Para representar la fecha 2 se tienen la variable `dd2` que corresponde al número del día de la fecha 2 y `mm2` que corresponde al número del mes de la fecha 2. Complete la línea 5 para que se obtenga como salida la diferencia en días entre ambas fechas.

Programa 2.27: Días transcurridos entre dos fechas en C

```
1 #include <stdio.h>
2
3 int main() {
4     int dd1=29, mm1=1;
5     int dd2=7, mm2=4;
6     int dias_de_diferencia = /*retirar comentario e incluir expresión solicitada*/;
7     printf("Entre %02d/%02d y %02d/%02d existen %d dias\n", dd1, mm1, dd2, mm2, dias_de_diferencia);
8     return 0;
9 }
```

Sugerencia

Asuma para este problema que el año en cuestión no es bisiesto y que todos los meses tienen 31 días. Asuma además que la fecha 1 será siempre menor o igual que la fecha 2.

Para poner en práctica

- Luego de completar el programa 2.27 y obtener la salida correcta, pruebe cambiando las fechas y verifique la salida obtenida.
- ¿Qué sucede cuando la fecha 1 es mayor que la fecha 2.
- Realice los cambios en el programa 2.27 de forma tal que permita representar la fecha conteniendo el año. Luego de representar el año, modifique la línea 5 para contar los días transcurridos.
- Calcule los días transcurridos desde su día de nacimiento hasta la fecha actual.
- Calcule los meses transcurridos desde su día de nacimiento hasta la fecha actual.
- Calcule los años transcurridos desde su día de nacimiento hasta la fecha actual.

2.3.10. Diagnóstico de enfermedades

En el mundo de la computación, los sistemas expertos intentan emular el comportamiento humano para realizar tareas que típicamente son realizadas por especialistas humanos. Una de esas tareas es el diagnóstico de enfermedades. Para realizar el diagnóstico el computador utiliza una serie de reglas: dado una serie de síntomas, intenta predecir si se tiene determinada enfermedad. Se desea realizar una expresión en C que dado una serie de síntomas definidos como proposiciones lógicas, retorne 1 si es que se posee determinada enfermedad o 0 en caso contrario. La enfermedad a diagnosticar en este caso es la peritonitis.

La peritonitis es la inflamación del peritoneo, una membrana suave que recubre las paredes abdominales internas y los órganos dentro del abdomen, la cual generalmente ocurre a causa de una infección bacteriana o micótica[4].

Entre sus síntomas[5] se tiene:

- Dolor abdominal o sensibilidad al tacto.
- Hinchazón o sensación de pesadez en el abdomen.
- Fiebre.
- Náuseas y vómitos.

Sugerencia

Realize un programa en C y represente en él cada proposición lógica como una variable. Luego en base a las variables definidas diseñe la expresión solicitada que permita determinar si se posee la enfermedad. Asuma que para que se posea la enfermedad, todas los síntomas deben estar presentes.

[4] Texto adaptado de la URL https://www.mayoclinic.org/es-es/diseases-conditions/peritonitis/symptoms-causes/syc-20376247

[5] Por simplicidad se presentan solamente 4 síntomas pero en la realidad esta enfermadad posee muchos más.

Referencias

[1] Steven Smith. *The scientist and engineerś guide to digital signal processing.* California Technical Pub. San Diego, 1997.