Pontificia Universidad Católica del Perú

Estudios Generales Ciencias

1INF01 - Fundamentos de Programación

Guía de laboratorio #1

Estructura básica de un programa

26 de agosto de 2018

Índice general

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.0	17.08.2018	A.Melgar	Versión inicial.

Siglas

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

Capítulo 1

Guía de Laboratorio #1

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 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.

1.2. Materiales y métodos

Como lenguaje de programación imperativo se utilizará el lenguaje ANSI C. Como Entorno de Desarrollo Integrado (IDE) para el lenguaje ANSI C se utilizará Dev C++[1]. No obstante, es posible utilizar otros IDEs como Netbeans y Eclipse.

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

[1] http://www.bloodshed.net/dev/devcpp.html
[2] http://dle.rae.es/?id=L2wAyos

la computadora debe realizar una tarea determinada. Hacer un programa en este paradigma imperativo significa básicamente escribir una lista de instrucciones que ordenen al computador la ejecución de determinadas 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 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, también en los laboratorios Bell desarrolló el 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 ANSI C. El ANSI C es un lenguaje estándar para C publicado por la American National Standards Institute (ANSI). Se creó para estandarizar las diversas implementaciones que existían para C en sus orígenes. Podrá encontrar una tarjeta de referencia del lenguaje ANSI C en español en la URL http://arantxa.ii.uam.es/~cantador/slides/tarjeta_referencia-ANSI_C.pdf y una tarjeta de ANSI C en inglés en la URL https://users.ece.utexas.edu/~adnan/c-refcard.pdf.

Programa 1.1: Primer programa en ANSI C

```
1  #include <stdio.h>
2
3  /*
4   * Este primer programa tiene la intencion de
5   * presentar la estructura basica de un programa
6   * usando el lenguaje ANSI C
7   */
8
9  int main() {
10     printf("Bienvenidos al curso de Fundamentos de Programacion");
11     return 0;
12 }
```

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

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 en la escritura de los 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 [**?**].

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 es considerado como 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 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 por su lado 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 previo 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 ANSI 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 `>`.`stdio.h` es un acrónimo de ***st**andard **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 ANSI 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 usar esta directiva de preprocesamiento.

1.3.2. Comentarios

Los comentarios son un elemento opcional en un programa en ANSI C, pero es recomendable colocarlos para explicar el objetivo de las rutinas. Esto facilita el entendimiento posterior del programa. En ANSI 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 ANSI C. Es la primera función que se ejecuta por lo que todo programa en ANSI 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 que tipo de resultado debe devolver. En la línea 9 del programa 1.1 se puede apreciar como se declara la función `main` como **`int`** `main()`. `int` es el tipo de resultado que se espera devolver, en este caso un entero. Los paréntesis vacios () indican que la función `main` no tendrá argumentos.

Para delimitar el bloque de instrucciones que se ejecutarán dentro de la función `main` se utilizan las llaves. En ANSI 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 sentencia `return 0`. Esta sentencia 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 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 del ANSI 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 este corresponde a `stdio`.

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 `"..."`). 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 ANSI 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` que `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 10 por `Return 0;` y verifique si el programa compila.

Luego de realizar las 3 alteraciones propuestas, ¿Qué puede decir del lenguaje ANSI C?

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 ANSI C, estos estados se representan mediante 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 las variables es que el valor que se almacena en ella puede variar a lo largo de un programa.

Para poder declarar una variable en ANSI 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 ANSI C básicamente trabaja con números. Existen varios tipos de datos en ANSI 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 $[-2,147,483,648, +2,147,483,647]$.

Tabla 1.1: Tipos de datos en ANSI C.

Tipo de dato	Descripción	Formato
char	Número entero (tipo de dato más pequeño en C).	%c
int	Número entero.	%d
double	Número real de doble presición.	%lf

Para poner en práctica

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

- Incluye el archivo de cabecera `limits` en la sección de directivas del preprocesador.
- Incluye 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.

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 ANSI C

Declaración

La declaración permitirá asignar el espacio de memoria que utilizará cada variable declarada. La forma general para la declaración de variables es como sigue:

```
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 1.2 se puede apreciar unos ejemplos de declaración de variables

Programa 1.2: Ejemplo de declaración de variables en ANSI C

```
1        int numero, suma;
2        double promedio;
3        char sexo;
```

Regla para formar identificadores en ANSI C

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

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

1.7. Operadores de asignación

En el ANSI C existen símbolos especiales que permiten realizar operaciones específicas predefinidas usando argumentos (*input*). 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 ANSI 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 un solo operando se llaman 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 1.3) permite asignar el valor del operando a una determinada variable. Es un operador unario. El resultado que retorna el operador 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 y a traves de la

asignación. En muchas ocasiones se utiliza la asignación para poder inicializar las variables, sobre todos 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: Operadores de Asignación.

Operador	Descripción
=	Asignación

1.7.1. Asignación en la declaración de una variable

El lenguaje ANSI 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. Notesé que en el caso de la variable sexo se le asigna el caracter 'F' que debe estar entre comillas simples. Podrá visualizar un ejemplo de lo mencionado en el programa 1.3.

Programa 1.3: Inicialización de variables al momento de su declaración en ANSI C

```
1        int suma=0;
2        double promedio=0.0;
3        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 1.4.

Programa 1.4: Impresión de variables en ANSI C

```
1  #include <stdio.h>
2
3  int main() {
4      int suma=0;
5      double promedio=0.0;
6      char sexo='F';
7
8      printf("%d\n", suma);
9      printf("%lf\n", promedio);
10     printf("%c\n", sexo);
11     return 0;
12 }
```

Luego de ejecutar el programa 1.4 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 ANSI 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 posee 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 caracter en cuestión sino que ejecuta la impresión dependiendo del caracter que le sigue.

En el caso del `\n`, el ANSI 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 1.4? Analise 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 caracter que representa el código ASCII (American Standard Code for Information Interchange (ASCII))[3].

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 1.4, 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 caracter cuyo código ASCII está siendo representado en la variable `sexo`. Dicho caracter debe ir entre comillas simples.

Para poner en práctica

¿Qué pasaría si en el programa 1.4 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 al tipo de datos de la variable. 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.

Para poner en práctica

¿Qué pasaría si en el programa 1.4 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 ANSI 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 1.4 y encuentre el patrón de comportamiento.

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

1.7.2. Asignación luego de la declaración 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 actualizar variables 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 1.5 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 1.5: Asignación de valores en ANSI 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 ANSI 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 1.6 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 ANSI 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 1.6: Asignación múltiple de valores en ANSI 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 1.6, 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 la variable `a`, el segundo `%d` será reemplazado por la variable `b` y el tercer `%d` será reemplazado por 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 1.6 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?

1.8. Operadores aritméticos

Los operadores aritméticos en ANSI C implementan las operaciones básicas de la aritmética: suma, resta, multiplicación y división. Toma como operando valores numéricos y retornar 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 de división entera

En el ANSI 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 1.7 puede apreciar un ejemplo de suma de números enteros en ANSI C.

Programa 1.7: Suma de enteros en ANSI C

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

Para poner en práctica

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

- Altere la línea 6 del programa 1.7 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 de ANSI C.

Hay que poner especial cuidado en 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 1.8, en la línea 6 se aprecia una expresión matemática cuyo resultado excede el rango de representación del tipo de datos `int`.

Programa 1.8: Suma de enteros en ANSI 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 enteros es de 2. Con 2 bits se pueden representar a lo más 4 números. ¿Qué pasaría si quisieramos 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 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 ANSI C para diferenciar si se debe realizar una división entera o real? Hace la diferenciación por los operando. Si los operandos son números enteros, entonces realiza una división entera. Si al menos un operando es real, realiza una división real.

Programa 1.9: División de números enteros en ANSI C

```
1 #include <stdio.h>
2
3 int main() {
4     int cociente, resto;
5
6     cociente = 5/2;
7     resto = 5%2;
8     printf("cociente = %d\n", cociente);
9     printf("resto = %d\n", resto);
10    return 0;
11 }
```

Luego de ejecutar el programa 1.9 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 1.10 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 1.10: División de números reales en ANSI 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 1.10?
- ¿Qué diferencia existen en la salida del programa 1.9 con salida del programa 1.10?
- ¿Qué pasaría si cambiásemos la línea 6 del programa 1.10 por `cociente = 5.0/2;`?

¿Cómo se hace para indicarle al programa en ANSI 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 1.11 se puede aprecir un ejemplo de conversión explícita de datos en la línea 6.

Programa 1.11: Conversión de tipo en división de números reales en ANSI 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 1.11?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.11 por `cociente=5/(double)2;`?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.11 por `cociente=1.0*5/2;`?
- ¿Qué pasaría si se cambiase la línea 6 del programa 1.11 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 ejecuan de izquierda a derecha.

Para poner en práctica

¿Qué resultado se obtiene luego de ejecutar el programa 1.12?

Programa 1.12: Precedencia de los operadores aritméticos en ANSI C

```
1 #include <stdio.h>
2
3 int main() {
4     int a=2, b=4, c=16, d;
5
6     d=c/b/a;
7     printf("%d\n", d);
8     return 0;
9 }
```

1.9. Operadores relacionales

Los operadores relacionales se utilizan para comparar dos valores. En ANSI C retornarán el valor de 1 si la comparación es correcta y 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. La expresiones relacionales se evaluan de izquierda a derecha. En la tabla 1.5 se pueden apreciar los operadores relaciones en ANSI C.

Tabla 1.5: Operadores relacionales.

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

Un error recurrente al momento del aprendizaje del lenguaje de programación ANSI C es confundir el operador de asignación (`=`) con el operador de comparación de igualdad (`==`). Ambos son operadores, retornan el mismo

tipo de resultado y por lo tanto pueden ser usado en diversas expresiones por lo que el lenguaje C no lo considera error.

Programa 1.13: Comparación de valores en ANSI 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 1.13 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 ANSI C una expresión lógica asume el valor de *verdadero* cuando esta tiene el valor diferente de 0 (tipicamente 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 ANSI C.

Tabla 1.6: Operadores lógicos.

Operador	Descripción
&&	Conjunción
||	Disyunción
!	Negación

En el programa 1.14 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 1.14: Conjunción en ANSI C

```
1 #include <stdio.h>
2
3 int main() {
4     int p, q, resultado;
5
6     p=1;
7     q=0;
8     resultado = p && q;
9     printf("resultado = %d\n", resultado);
10    return 0;
11 }
```

Para poner en práctica

- Altere los valores de `p` y `q` en el programa 1.14 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 1.14 para probar la disyunción. Use el operador `||`.
- Tomando como base el programa 1.14 implemente la condicional $(p \rightarrow q)$.

  Recuerde que $(p \rightarrow q \equiv \neg p \vee q)$
- Tomando como base el programa 1.14 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 1.15). 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, en ANSI 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 1.15: Expresiones complejas en ANSI C

```
1  #include <stdio.h>
2
3  int main() {
4      int a, b, n, resultado;
5
6      a=15;
7      b=20;
8      n=101;
9      resultado = a<b && n<=100;
10     printf("resultado = %d\n", resultado);
11     return 0;
12 }
```

1.11. Ejercicios propuestos

1.11.1. Precedencia de operadores

Para cada uno de las siguientes expresiones, se solicita que elabore un programa en ANSI C que verifique el valor resultante de cada expresión.

- `5+3*2+7`
- `(5+3)*(2+7)`

1.11.2. Expresiones lógicas

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

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

1.11.3. Conversión de grados centígrados en Fahrenheit

El programa 1.16 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 1.16: Conversión de grados centígrados en Fahrenheit en ANSI 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 }
```