Tipos de Datos Primarios

En este capítulo se estudian los conceptos básicos de la memoria de la computadora, así como la forma de obtener información de los usuarios y almacenarla como datos mediante tipos de datos del lenguaje C. Además de los tipos de datos básicos, también aprenderá a mostrar el contenido de las variables mediante la función `printf()` y a manipular los datos almacenados en las variables mediante aritmética básica.

Conceptos de memoria

La memoria de una computadora es similar a la de un ser humano, ya que tiene memoria a corto y largo plazo. La memoria a largo plazo de una computadora se denomina memoria *no volátil* y generalmente se asocia con dispositivos de almacenamiento masivo, como discos duros, matrices de discos grandes, almacenamiento óptico (CD/DVD) y, por supuesto, dispositivos de almacenamiento portátiles como memorias USB o unidades flash. En los capítulos 10 y 11, aprenderá a utilizar la memoria no volátil para almacenar datos.

Este capítulo se concentra en la memoria a corto plazo o *volátil* de una computadora. La memoria volátil pierde sus datos cuando se corta la energía de la computadora. Se la conoce comúnmente como RAM (memoria de acceso aleatorio).

La RAM está compuesta por celdas de tamaño fijo y cada número de celda se referencia a través de una dirección. Los programadores suelen hacer referencia a las celdas de memoria mediante el uso de variables. Hay muchos tipos de variables, según el lenguaje de programación, pero todas comparten características similares, como se describe en la Tabla 2.1.

Tabla 2.1: Características comunes de las variables

Atributo de la variable	Descripción
Nombre	El nombre de la variable que se utiliza para hacer referencia a los datos en el código del programa
Tipo	El tipo de datos de la variable (número, carácter, etc.)
Valor	El valor de los datos asignado a la ubicación de la memoria
Dirección	La dirección asignada a una variable, que apunta a la ubicación de una celda de memoria

Utilizando los atributos definidos en la Tabla 2.1, la Figura 2.1 muestra la relación gráfica para algunos tipos de datos comunes. Observe que las letras y los números en la columna "Dirección de memoria" en la Figura 2.1, como FFF4, representan ubicaciones de memoria en el sistema de numeración hexadecimal. El sistema de numeración hexadecimal se utiliza a veces en la programación avanzada en C para hacer referencia a direcciones de memoria concisas, como durante la programación a nivel de sistema.

Nombre de variable	Valor	Tipo	Dirección de memoria
Openard1	29	int	FFF4
Resultado	756.21	float	FHH6
Inicial	M	char	FHF2

Figura 2.1: Representación de atributos de variable comunes y valores de muestra.

Tipos de Datos

Descubrirá muchos tipos de datos en su carrera de programación, como números, fechas, cadenas, booleanos, matrices, objetos y estructuras de datos. Aunque este libro cubre algunos de los tipos de datos mencionados anteriormente en capítulos posteriores, este capítulo se concentrará en los siguientes tipos de datos principales:

- Integers
- Floating-point numbers
- Characters

Enteros

Los números enteros son números naturales que representan números positivos y negativos, como -3, -2, -1, 0, 1, 2 y 3, pero no números decimales ni fraccionarios.

Los tipos de datos enteros contienen un máximo de cuatro bytes de información y se declaran con la palabra clave `int` (abreviatura de integer/entero), como se muestra en la siguiente línea de código.

```
int x;
```

En C, puede declarar más de una variable en la misma línea utilizando una única declaración `int` con cada nombre de variable separado por comas, como se muestra a continuación.

```
int x, y, z;
```

La declaración de variable anterior declara tres variables enteras denominadas `x`, `y` y `z`. Recuerde del Capítulo 1 que las instrucciones de programas ejecutables, como una instrucción de impresión o, en este caso, una declaración de variable, requieren un terminador de instrucción (`;`).

Números de punto flotante

Los *números de punto flotante* son todos los números, incluidos los decimales y fraccionarios con y sin signo. Los números con signo incluyen números positivos y negativos, mientras que los números sin signo solo pueden incluir valores positivos. En la siguiente lista se muestran ejemplos de números de punto flotante.

- 09.4543
- 3428.27
- 112.34329
- -342.66
- -55433.33281

Utilice la palabra clave `float` para declarar números de punto flotante, como se muestra a continuación.

```
float operando1;
float operando2;
float resultado;
```

El código anterior declara tres tipos de datos de variables de punto flotante denominados `operando1`, `operando2` y `resultado`.

Caracteres

Los tipos de datos de caracteres son representaciones de valores enteros conocidos como *códigos de caracteres*. Por ejemplo, el código de carácter 90 representa la letra Z. Tenga en cuenta que la letra Z no es lo mismo que el código de carácter 122, que representa la letra z (letra z minúscula).

Los caracteres representan más que sólo las letras del alfabeto; también representan números del 0 al 9, caracteres especiales como el asterisco (*) y teclas del teclado como la tecla Del (borrar) y la tecla Esc (escape). En total, hay un total de 128 códigos de caracteres comunes (del 0 al 127), que conforman los caracteres más utilizados de un teclado.

Los códigos de caracteres se organizan principalmente a través del conjunto de caracteres ASCII (*American Standard Code for Information Interchange*). Para obtener una lista de códigos de caracteres ASCII comunes, consulte el Apéndice D, "Códigos de caracteres ASCII comunes".

> **ASCII**
>
> ASCII o Código Estándar Americano para el Intercambio de Información (*American Standard Code for Information Interchange*) se destaca por su conjunto de caracteres, que utiliza valores enteros pequeños para representar caracteres o valores del teclado.

En C, las variables de caracteres se crean utilizando la palabra clave char (abreviatura de carácter), como se muestra a continuación.

```
char firstInitial;
char middleInitial;
char lastInitial;
```

Los datos de caracteres asignados a las variables de caracteres deben estar entre comillas simples ('), también conocidas como marcas de graduación o apóstrofos. Como verá en la siguiente sección, el signo igual (=) se utiliza para asignar datos a la variable de caracteres.

Precaución

No puede asignar varios caracteres a un único tipo de variable de caracteres. Cuando se necesita más de un carácter para almacenar una única variable, debe utilizar una matriz de caracteres (que se analiza en el Capítulo 6, "Matrices") o cadenas (que se analizan en el Capítulo 8, "Cadenas").

Inicialización de variables y el operador de asignación

Cuando se declaran las variables por primera vez, el programa asigna el nombre de la variable (puntero de dirección) a una ubicación de memoria disponible. Nunca es seguro asumir que la ubicación de la variable recién asignada está vacía. Es posible que la ubicación de memoria contenga datos utilizados anteriormente (o basura). Para evitar que aparezcan datos no deseados en las variables recién creadas, inicialice las nuevas variables, como se muestra a continuación.

```
/* Declare variables */
int x;
char firstInitial;

/* Initialize variables */
x = 0;
firstInitial = '\0';
```

El código anterior declara dos variables: una de tipo entero y otra de tipo carácter. Después de crear las dos variables, las inicializo con un valor particular. Para la variable entera, asigno el valor cero (0) y para el tipo de datos carácter, asigno el conjunto de caracteres \0, que se conoce como el carácter NULL.

Observe que en la asignación de datos de la variable carácter encerré el carácter `NULL` entre comillas simples. Las comillas simples son necesarias cuando se asignan datos al tipo de datos carácter.

El tipo de datos `NULL` se utiliza comúnmente para inicializar ubicaciones de memoria en lenguajes de programación, como C, y bases de datos relacionales, como Oracle y SQL Server.

Aunque los tipos de datos `NULL` son un concepto común en informática, pueden ser confusos. Básicamente, los caracteres `NULL` son tipos de datos desconocidos almacenados en una ubicación de memoria. Sin embargo, no es adecuado pensar en los datos `NULL` como vacíos o nulos; en cambio, piense en los datos `NULL` como simplemente indefinidos.

Al asignar datos a variables, como la inicialización de variables, el signo igual no se utiliza en un sentido comparativo. En otras palabras, no dirías que `x` es igual a `0`. En cambio, los programadores dicen que la variable `x` toma el valor `0`.

Recuerda que, al asignar datos a variables, como al inicializar, te refieres al signo igual como un operador de asignación, no como un operador de comparación.

También puedes inicializar tus variables mientras las declaras, como se muestra a continuación.

```
int x = 0;
char firstInitial = '\0';
```

El código anterior realiza las mismas tareas en dos líneas que el código siguiente realiza en cuatro.

```
int x;
char firstInitial;
x = 0;
firstInitial = '\0';
```

Impresión del contenido de las variables

Para imprimir el contenido de las variables, utilice la función `printf()` con algunas opciones de formato nuevas, como se muestra en el siguiente bloque de código.

```
#include <stdio.h>

void main()
{
    // declaraciones de variables
    int x;
    float y;
    char c;

    // inicialización de variables
    x = -4443;
    y = 554.21;
    c = 'M';

    // Impresión de contenidos variables en la salida estándar
    printf("\nEl valor de la variable entera x es %d", x);
    printf("\nEl valor de la variable flotante y es %f", y);
    printf("\nEl valor de la variable de carácter c es %c\n", c);
}
```

Primero, declaro tres variables (un entero, un flotante y un carácter) y luego inicializo cada una de ellas. Después de inicializar las variables, uso la función `printf()` y los especificadores de conversión

(que se analizan a continuación) para mostrar el contenido de cada variable en la pantalla de la computadora.

El código anterior es un programa C completo que demuestra muchos de los temas tratados hasta ahora (su salida se muestra en la Figura 2.2).

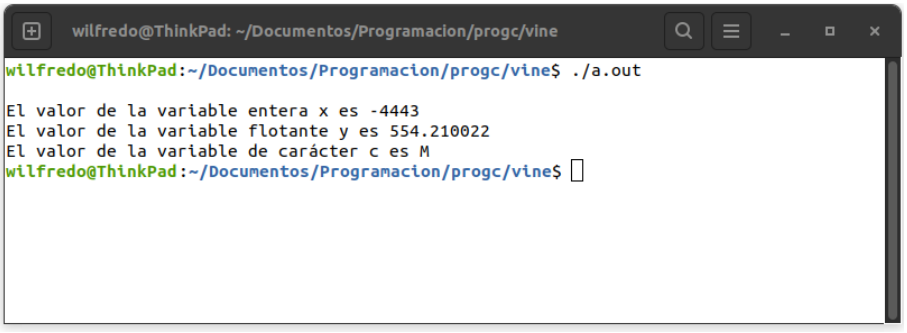

Figura 2.2: Impresión de contenidos variables.

Especificadores de conversión

Dado que la información se almacena como datos ilegibles en la memoria de la computadora, los programadores en C deben indicar específicamente a las funciones de entrada o salida, como `printf()`, cómo mostrar los datos como información. Puede lograr esta tarea aparentemente difícil utilizando conjuntos de caracteres conocidos como *especificadores de conversión*.

Los especificadores de conversión se componen de dos caracteres: el primer carácter es el signo de porcentaje (%) y el segundo es un carácter especial que le indica al programa cómo convertir los datos. La Tabla 2.2 describe los especificadores de conversión más comunes para los tipos de datos analizados en este capítulo.

Tabla 2.2: Especificadores de conversión comunes utilizados con `printf()`

Especificador de conversión	Descripción
%d	Muestra un valor entero
%f	Muestra números de punto flotante
%c	Muestra un carácter

Visualización de tipos de datos enteros con `printf()`

Los tipos de datos enteros se pueden visualizar fácilmente utilizando el especificador de conversión %d con una declaración `printf()` como se muestra a continuación.

```
printf("%d", 55);
```

La salida de la declaración anterior imprime el siguiente texto:

```
55
```

El especificador de conversión %d también se puede utilizar para generar el contenido de una variable declarada como tipo de datos entero, como se muestra a continuación.

```
int operando1;
operando1 = 29;
printf("El valor de operando1 es %d", operando1);
```

En las declaraciones anteriores, declaro una nueva variable entera llamada `operando1`. A continuación, asigno el número 29 a la variable recién creada y muestro su contenido utilizando la función `printf()` con el especificador de conversión %d.

Cada variable mostrada mediante una función `printf()` debe estar fuera de los paréntesis y separada por una coma (,).

Cómo visualizar tipos de datos de punto flotante con `printf()`

Para visualizar números de punto flotante, utilice el especificador de conversión %f que se muestra a continuación.

```
printf("%f", 55.55);
```

He aquí otro ejemplo del especificador de conversión %f, que imprime el contenido de una variable de punto flotante:

```
float result;
result = 3.123456;
printf("The value of result is %f", result);
```

Aunque el especificador de conversión %f muestra números de punto flotante, puede que no sea suficiente para mostrar el número de punto flotante con la precisión correcta o deseada. La siguiente función `printf()` demuestra el problema de precisión.

```
printf("%f", 55.55);
```

Este ejemplo de `printf()` genera un número de punto flotante con una precisión de seis dígitos a la derecha del punto decimal, como se muestra a continuación.

```
55.550000
```

Para crear precisión con números de punto flotante, ajuste el especificador de conversión utilizando esquemas de numeración entre el signo % y el especificador de conversión de carácter f.

```
printf("%.1f", 3.123456);
printf("\n%.2f", 3.123456);
printf("\n%.3f", 3.123456);
printf("\n%.4f", 3.123456);
printf("\n%.5f", 3.123456);
printf("\n%.6f", 3.123456);
```

El bloque de código anterior produce el siguiente resultado:

```
3.1
3.12
3.123
3.1234
3.12345
3.123456
```

Tenga en cuenta que he incluido la secuencia de escape $\backslash n$ en cada una de las sentencias de impresión anteriores (excepto la primera línea de código). Sin la secuencia de escape de nueva línea ($\backslash n$), la salida de cada sentencia se generaría en la misma línea, lo que dificultaría su lectura.

Visualización de tipos de datos de caracteres con `printf()`

Los caracteres también son fáciles de visualizar utilizando el especificador de conversión %c.

```
printf("%c", 'M');
```

La salida de esta declaración es simplemente la letra M. Al igual que los otros especificadores de conversión, puede generar el contenido de un tipo de datos de variable de carácter utilizando el especificador de conversión %c y una función `printf()` como se muestra a continuación.

```
char firstInitial;
firstInitial = 'S';
printf("The value of firstInitial is %c", firstInitial);
```

Puede utilizar múltiples especificadores de conversión en una sola función `printf()`

```
char firstInitial, middleInitial, lastInitial;
firstInitial = 'M';
middleInitial = 'A';
lastInitial = 'V';
printf("My Initials are %c.%c.%c.", firstInitial, middleInitial, lastInitial);
```

La salida de las sentencias del programa anterior es la siguiente.

```
My Initials are M.A.V.
```

Observe en la declaración a continuación que cada variable que se muestra con la función `printf()` está fuera de las comillas dobles y separada por una sola coma.

```
printf("My Initials are %c.%c.%c.", firstInitial, middleInitial, lastInitial);
```

El texto dentro de las comillas dobles de `printf()` está reservado para texto visualizable, especificadores de conversión y secuencias de escape.

Constantes

Los tipos de datos constantes, a los que a menudo se denomina variables de solo lectura, no pueden perder sus valores durante la ejecución del programa. Se utilizan con mayor frecuencia cuando es necesario reutilizar un valor de datos común sin modificarlo.

Los valores de datos constantes pueden ser de muchos tipos, pero se deben asignar cuando se crea la constante por primera vez, como se muestra a continuación.

```
const int x = 20;
const float PI = 3.14;
```

Tenga en cuenta que la palabra clave `const` precede al nombre del tipo de datos, lo que indica que se trata de una variable o constante de solo lectura. Puede imprimir los valores de las constantes de la misma manera que se imprimen las variables normales utilizando especificadores de conversión con la función `printf()`, como se muestra en el siguiente código de programa:

```
#include <stdio.h>

void main()
```

```
{
    const int x = 20;
    const float PI = 3.14;

    printf("\nLos valores constantes son %d y %.2f\n", x, PI);
}
```

La figura 2.3 muestra la salida del bloque de código anterior.

Figura 2.3: Impresión de valores de tipo de datos constantes.

Convenciones y estilos de programación

Si nadie te ha mencionado esto todavía, déjame ser el primero en decirte que la programación es tanto un arte como una ciencia. Tus programas son un reflejo de ti y deben revelar un estilo fluido y consistente que guíe la mirada del lector a través de algoritmos y flujo de programas. Así como un puente proporciona una función, también puede proporcionar belleza, un placer visual tanto para el ingeniero estructural como para el viajero.

Debes ceñirte a un estilo y una convención que te permitan a ti o a otra persona leer fácilmente tu código. Una vez que elijas o te sientas cómodo con un estilo de programación, el objetivo es la coherencia. En otras palabras, apégate a él, no mezcles convenciones de nombres para variables ni entremezcles estilos de sangría dentro del mismo programa.

Al aprender a programar, debes considerar específicamente al menos dos áreas para desarrollar una convención y un estilo de programación consistentes.

- Espacio en blanco
- Convenciones de nombres de variables

Espacio en blanco

No se suele hablar de espacio en blanco en los círculos de programación, ya que no ofrece ningún beneficio informático. De hecho, el compilador ignora el espacio en blanco, por lo que eres libre de tratarlo como quieras. Entonces, ¿qué es el espacio en blanco? Filosóficamente hablando, el espacio en blanco es tu lienzo de programación. Si se utiliza mal, puede cansar la vista del lector; si se pinta correctamente, puede ser beneficioso. Algunos ejemplos de cómo se puede controlar el espacio en blanco son las llaves y la sangría.

La sangría es imprescindible, ya que guía a tus ojos dentro y fuera del control del programa. Por ejemplo, al observar la siguiente función de ejemplo main(), tus ojos te dicen rápidamente que el código dentro de la función pertenece lógicamente a ella.

```
main()
{
    //you code in here
}
```

Una discusión común sobre la sangría es el viejo argumento de tabulaciones versus espacios. Este argumento se puede resolver con bastante facilidad a favor de los espacios. La razón detrás de esta preferencia se basa en el hecho de que las tabulaciones se pueden configurar para ocupar varias columnas. Otro programador que abra su código podría no tener la misma cantidad de columnas configuradas para sus tabulaciones y, en consecuencia, el formato estará desfasado.

Otra pregunta común entre los programadores principiantes es cuánto sangrar. Personalmente, prefiero una sangría de dos a cuatro espacios. Una sangría de más de cuatro espacios eventualmente conducirá a líneas que sean demasiado largas. El objetivo aquí es mantener un estilo de sangría consistente que mantenga las líneas de código en la pantalla de la computadora.

Otra cosa a considerar con respecto al espacio en blanco son los estilos de llaves, que están estrechamente relacionados con su estilo de sangría. Al igual que con la sangría, hay varios estilos de llaves, aunque probablemente prefiera uno de ellos

```
main()
{
    //you code in here
}
```

o este

```
main(){
    //your code in here
}
```

Como con cualquier estilo, la elección es suya, aunque le recomiendo equilibrar un estilo que sea cómodo para usted y que sea consistente con lo que usan los demás en su equipo.

Convenciones de nomenclatura de variables

La siguiente lista contiene una cantidad mínima de pautas que debe seguir al declarar y nombrar sus variables.

- Identifique los tipos de datos con un prefijo.
- Use letras mayúsculas y minúsculas de manera adecuada.
- Dé a las variables nombres significativos.

No existe una única forma correcta de implementar una nomenclatura para los nombres de las variables, aunque algunas son mejores que otras. Después de identificar sus estándares de nomenclatura, el proceso más importante es mantener la coherencia con esas prácticas en todos sus programas.

En las siguientes secciones, le mostraré un par de formas diferentes que han funcionado para mí y para muchos otros programadores que han utilizado las pautas de la lista anterior.

Trampa

Además de respetar una convención de nombres de variables, tenga cuidado de no utilizar caracteres reservados en los nombres de las variables. Como regla general, respete las siguientes sugerencias:

- Siempre comience los nombres de las variables con una letra minúscula.
- No utilice espacios en los nombres de las variables.
- Utilice únicamente letras, números y guiones bajos (_) en los nombres de las variables.
- Mantenga los nombres de las variables con menos de 31 caracteres para cumplir con los estándares ANSI C.

Identificación de tipos de datos con un prefijo

Cuando trabajo con variables, suelo elegir uno de los tres tipos de prefijos, como se muestra a continuación.

```
int intOperand1;
float fltResult;
char chrMiddleInitial;
```

Para cada tipo de datos de variable, elijo un prefijo de tres caracteres, `int` (abreviatura de entero), `flt` (abreviatura de float) y `chr` (abreviatura de character), para los prefijos de los nombres de las variables. Cuando veo estas variables en el código de mi programa, sé al instante qué tipos de datos son.

Otra forma de prefijar los tipos de datos de enteros es usar un prefijo de un solo carácter, como se muestra en las segundas declaraciones de variable.

```
int iOperand1;
float fResult;
char cMiddleInitial;
```

Aunque estas variables no indican claramente su tipo de datos, puedes ver su prefijo fácilmente al intentar determinar el tipo de contenido de la variable. Además, estos prefijos de un solo carácter funcionan muy bien cuando se utilizan junto con letras mayúsculas y minúsculas adecuadas, como se explica en la siguiente sección.

Uso adecuado de letras mayúsculas y minúsculas

La convención de nombres de variables más común y preferida es escribir con mayúscula el primer carácter de cada palabra en el nombre de una variable (como se muestra en el código siguiente).

```
float fNetSalary;
char cMenuSelection;
int iBikeInventoryTotal;
```

El uso de caracteres en mayúsculas en cada palabra facilita la lectura del nombre de la variable y la identificación de su propósito. Ahora, observe las mismas variables con el mismo nombre, solo que esta vez sin utilizar caracteres en mayúsculas.

```
float fnetsalary;
char cmenuselection;
int ibikeinventorytotal;
```

¿Qué nombres de variables son más fáciles de leer?

Además de utilizar letras mayúsculas para facilitar la lectura, a algunos programadores les gusta utilizar el carácter de subrayado para separar las palabras, como se muestra en el siguiente código.

```
float f_Net_Salary;
char c_Menu_Selection;
int i_Bike_Inventory_Total;
```

El uso del carácter de subrayado crea una variable legible, pero para mí es un poco complicado.

Los tipos de datos constantes plantean otro desafío para la creación de una convención de nombres estándar. Personalmente, me gustan las siguientes convenciones de nombres.

```
const intconstWeeks = 52;
const intWEEKS = 52;
```

En la primera declaración de constante, utilizo el prefijo `const` para identificar a `constWeeks` como una constante. Sin embargo, observe que aún escribo en mayúscula la primera letra del nombre de la constante para facilitar su lectura.

En la segunda declaración, simplemente escribo en mayúscula cada letra del nombre de la constante. Este estilo de denominación realmente se destaca.

Dar nombres significativos a las variables

Darles nombres significativos a las variables es probablemente la parte más importante de las convenciones de nombres de variables. Al hacerlo, se crea un código autodocumentado. Considere la siguiente sección de código, que utiliza comentarios para describir el propósito de la variable.

```
int x; //x es la edad
int y; //y es la distancia
int z; //z es el resultado
```

Las declaraciones de variables anteriores no utilizan nombres significativos y, por lo tanto, requieren algún tipo de documentación para que el propósito de su código sea comprensible. En su lugar, observe los siguientes nombres de variables que se documentan por sí mismos.

```
int iEdad;
int iDistancia;
int iResultado;
```

scanf()

Hasta ahora, ha aprendido a enviar la salida a la pantalla de la computadora mediante la función `printf()`. En esta sección, aprenderá a recibir la entrada de los usuarios a través de la función `scanf()`.

La función `scanf()` es otra función incorporada proporcionada por la biblioteca de entrada y salida estándar `<stdio.h>`; lee la entrada estándar del teclado y la almacena en variables declaradas previamente. Toma dos argumentos, como se muestra a continuación.

```
scanf("especificador de conversión", variable);
```

El argumento del *especificador de conversión* le dice a `scanf()` cómo convertir los datos entrantes. Puede utilizar los mismos especificadores de conversión que se analizaron en la Tabla 2.2 y se muestran nuevamente en relación con `scanf()` en la Tabla 2.3.

Tabla 2.3: Especificadores de conversión comunes utilizados con `scanf()`

Especificador de conversión	Descripción
%d	Recibe un valor entero
%f	Recibe números de punto flotante
%c	Recibe caracteres

El código siguiente representa un programa C completo, el programa Sumador, que utiliza la función `scanf()` para leer dos números enteros y sumarlos. Su salida se muestra en la Figura 2.4.

```
#include <stdio.h>

void main()
{
    int iOperando1 = 0;
    int iOperando2 = 0;

    printf("\n\tPrograma Sumador, por Michael Vine\n");
    printf("\nEntre primer Operando: ");
    scanf("%d", &iOperando1);
    printf("Entre segundo Operando: ");
    scanf("%d", &iOperando2);
    printf("El resultado es %d\n", iOperando1 + iOperando2);
}
```

Figura 2.4: Uso de `scanf()` para recibir entrada de un usuario.

La primera línea de código notable solicita al usuario que ingrese un número.

```
printf("\nIngrese el primer operando: ");
```

Puede notar que la función `printf` anterior no contiene una variable al final, ni incluye la secuencia de escape $\backslash n$ al final de la instrucción. Al dejar la secuencia de escape de nueva línea fuera del final de una instrucción `printf`, el control del programa se detiene mientras espera la entrada del usuario.

La siguiente línea de código utiliza la función `scanf()` para recibir la entrada del usuario.

```
scanf("%d", &iOperando1);
```

El primer argumento `scanf()` toma el especificador de conversión de enteros ("%d"), que le indica al programa que convierta el valor entrante en un entero. El segundo operador es un operador de dirección (&), seguido del nombre de la variable.

Básicamente, el operador de dirección contiene un puntero a la ubicación en la memoria donde se encuentra su variable. Aprenderá más sobre el operador de dirección (&) en el Capítulo 7, cuando analice los punteros. Por ahora, solo debes tener en cuenta que debes preceder los nombres de las variables con este símbolo cuando uses la función `scanf()`.

Precaución

Olvidar colocar el operador de dirección (`&`) delante de la variable en una función `scanf()` *no siempre generará errores de compilación*, pero causará problemas con el acceso a la memoria durante la ejecución del programa.

Después de recibir ambos números (operandos) del usuario, utilizo una declaración de impresión para mostrar el siguiente resultado.

```
printf("El resultado es %d\n", iOperando1 + iOperando2);
```

En esta declaración de impresión, incluyo un único especificador de conversión (`%d`), que le indica al programa que muestre un único valor entero. En el siguiente argumento de la función `printf()`, sumo ambos números ingresados por el usuario utilizando el signo de adición (+).

Aritmética en C

Como se demostró en el programa Sumador de la sección anterior, C permite a los programadores realizar todo tipo de operaciones aritméticas. La Tabla 2.4 muestra los operadores aritméticos más comunes que se utilizan en la programación inicial en C.

Tabla 2.4: Operadores aritméticos comunes

Operador	Descripción	Ejemplo
*	Multiplicación	fResultado = fOperando1 * fOperando2;
/	División	fResultado = fOperando1 / fOperando2;
%	Módulo (resto)	fResto = fOperando1 % fOperando2;
+	Suma	fResultado = fOperando1 + fOperando2;
-	Resta	fResultado = fOperando1 - fOperando2;

En el programa Sumador de la sección anterior, utilicé un atajo al trabajar con operaciones aritméticas comunes: realicé mi cálculo en la función `printf()`. Aunque esto no es obligatorio, puede utilizar variables y sentencias de programa adicionales para obtener el mismo resultado. Por ejemplo, el siguiente código es otra variación del programa Sumador que utiliza sentencias de programa adicionales para lograr el mismo resultado.

```
#include <stdio.h>

void main()
{
    int iOperando1 = 0;
    int iOperando2 = 0;
    int iResultado = 0;

    printf("\n\tPrograma Sumador, por Michael Vine\n");
    printf("\nEntre primer operando: ");
    scanf("%d", &iOperando1);
    printf("Entre segundo operando: ");
    scanf("%d", &iOperando2);

    iResultado = iOperando1 + iOperando2;

    printf("El resultado es %d\n", iResultado);
}
```

En esta desviación del programa Sumador, utilicé dos instrucciones adicionales para obtener el mismo resultado. En lugar de realizar la aritmética en la función `printf()`, declaré una variable adicional llamada `iResult` y le asigné el resultado de `iOperand1 + iOperand2` mediante una instrucción separada, como se muestra a continuación.

```
iResult = iOperand1 + iOperand2;
```

Recuerde que el signo igual (=) es un operador de asignación, donde el lado derecho se asigna al lado izquierdo del operador (=). Por ejemplo, no diría lo siguiente:

```
iResult es igual a iOperand1 más iOperand2.
```

Eso está mal expresado. En su lugar, diría:

```
iResult obtiene el valor de iOperand1 más iOperand2.
```

Precedencia de operadores

La precedencia de operadores es muy importante cuando se trabaja con operaciones aritméticas en cualquier lenguaje de programación. La precedencia de operadores en C se muestra en la Tabla 2.5.

Tabla 2.5: Precedencia de operadores

Orden o precedencia	Descripción
()	Los paréntesis se evalúan primero, desde el más interno al más externo
*, /, %	Se evalúan en segundo lugar, de izquierda a derecha
+,-	Se evalúan en último lugar, de izquierda a derecha

Tomemos la siguiente fórmula, por ejemplo, que utiliza paréntesis para dictar el orden adecuado de las operaciones.

```
f = (a - b)(x - y);
```

Dado `a = 5`, `b = 1`, `x = 10` e `y = 5`, puede implementar la fórmula en C utilizando la siguiente sintaxis.

```
intF = (5 - 1) * (10 - 5);
```

Si se utiliza el orden correcto de operaciones, el valor de `intF` sería `20`. Observe nuevamente la misma implementación en C, esta vez sin utilizar paréntesis para indicar el orden correcto de operaciones.

```
intF = 5 - 1 * 10 - 5;
```

Si no se implementa el orden correcto de operaciones, `intF` daría como resultado `-10`.

Programa de capítulos: Profit Wiz

Como se muestra en la Figura 2.5, el programa **Profit Wiz** utiliza muchos conceptos basados en capítulos, como variables, entrada y salida con funciones `printf()` y `scanf()`, y aritmética básica.

Figura 2.5: Demostración de conceptos basados en capítulos con el programa Profit Wiz

A continuación se muestra todo el código C necesario para crear el programa Profit Wiz.

```
#include <stdio.h>

void main()
{
float fGanancia, fCosto;

fGanancia = 0;
fCosto = 0;

/* beneficio = Ganancia - Costo */

printf("\nEnter total Ganancia: ");
scanf("%f", &fGanancia);
printf("\nEnter total costo: ");
scanf("%f", &fCosto);
printf("\nSu beneficio es $%.2f\n", fGanancia - fCosto);
}
```

Resumen

- La memoria a largo plazo de una computadora se denomina memoria *no volátil* y generalmente se asocia con dispositivos de almacenamiento masivo, como discos duros, matrices de discos grandes, disquetes y CD-ROM.
- La memoria a corto plazo de una computadora se denomina memoria *volátil* y pierde sus datos cuando se corta la energía de la computadora.
- Los números enteros son números enteros que representan números positivos y negativos.
- Los números de punto flotante representan todos los números, incluidos los números decimales y fraccionarios con y sin signo.
- Los números con signo incluyen números positivos y negativos, mientras que los números sin signo solo pueden incluir valores positivos.
- Los tipos de datos de caracteres son representaciones de valores enteros conocidos como códigos de caracteres.
- Los especificadores de conversión se utilizan para mostrar datos ilegibles en la memoria de una computadora como información.

- Los tipos de datos constantes conservan sus valores de datos durante la ejecución del programa.
- Los compiladores ignoran los espacios en blanco y, por lo general, se administran para facilitar la lectura mediante estilos de programación como sangría y colocación de llaves.
- Tres reglas útiles para las convenciones de nombres incluyen:
    1. Identificar los tipos de datos con un prefijo.
    2. Use letras mayúsculas y minúsculas de forma adecuada.
    3. Dé a las variables nombres significativos.
- La función `scanf()` lee la entrada estándar del teclado y la almacena en variables declaradas previamente.
- El signo igual (=) es un operador de asignación, donde el lado derecho del operador de asignación se asigna al lado izquierdo del operador.
- En la precedencia de operadores, los paréntesis se evalúan primero, desde el más interno hasta el más externo.