Conceptos Básicos de Programación

Domina Cualquier Lenguaje

Conceptos Básicos de Programación - Domina Cualquier Lenguaje

INSTRUCTOR

Marco Mendoza

Ingeniero en Sistemas, Téc en Informática y Autodidacta

Sobre mí

Soy ingeniero en sistemas, técnico en informática y autodidacta, a lo largo de mi vida he tenido la oportunidad de trabajar en diferentes campos como la administración de sistemas, diseño y programación web, desarrollo de aplicaciones móviles y también como maestro. hoy en día con el conocimiento que gane por mis años de trabajo, más muchas horas de estudio autodidacta, he llegado a desempeñarme en el área de seguridad informática realizando auditorias de seguridad en diferentes empresas e instituciones de gobierno.

También soy el creador de una pequeña comunidad en YouTube llamada Hacking y Más donde imparto algunos cursos de seguridad y hacking ético.

Redes sociales:

Página Web

Facebook

YouTube

Telegram

Instagram

Twitter

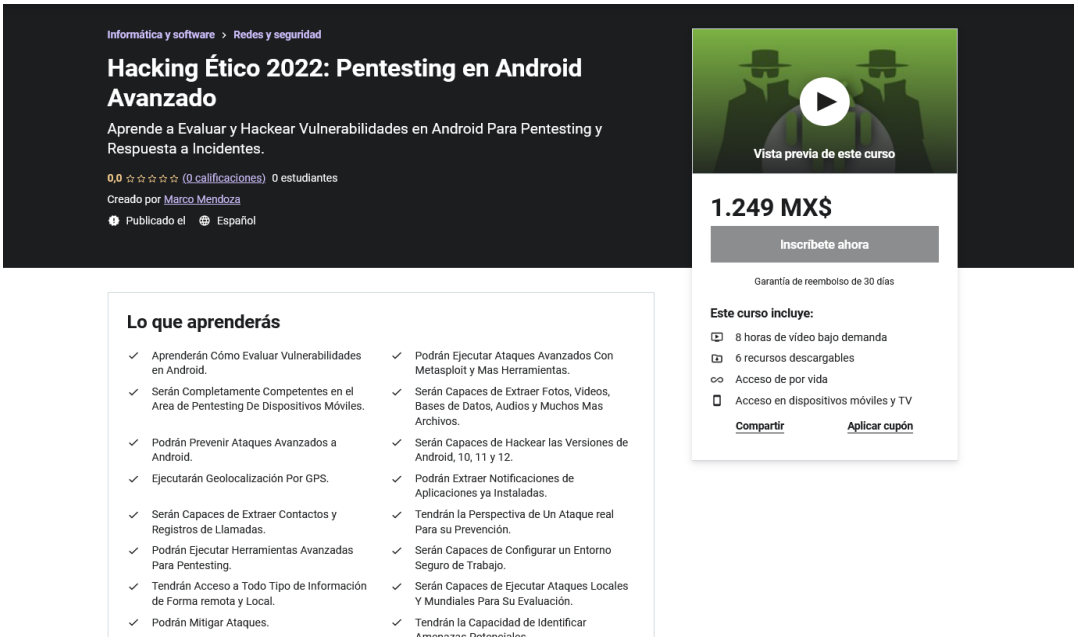
Descuentos en los mejores cursos

Hacking Ético 2022: Pentesting en Android Avanzado

Si quieres aprender como hackear Android 11, 12, 13 y extraer fotos, videos, notificaciones de redes sociales, posición GPS, contactos, SMS, Bases de datos de Whatsapp y tener a disposición cualquier información, este es tu curso.

CUPON DEL 80% DE DESCUENTO AQUÍ EN MI PAGINA WEB: https://bit.ly/3JcdOlw

Únete a mis **62.139** estudiantes de **Udemy** y aprende conmigo.



Descripción

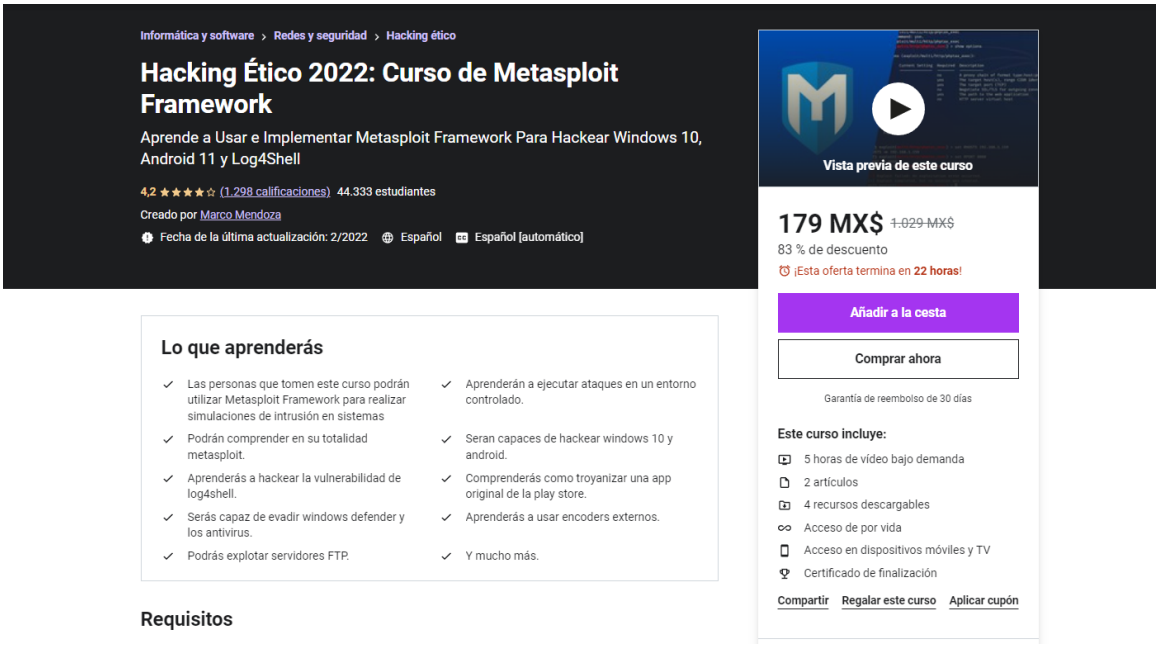
La seguridad en dispositivos **Android** se ha vuelto cada vez más importante debido a la gran demanda que hay en el mundo con respecto a Smartphones, Smart tv, relojes, impresoras e incluso computadoras y laptops. Es debido a este incremento de dispositivos inteligentes que usan **Android** que las amenazas para los usuarios han aumentado de una manera desproporcionada, un claro ejemplo de esto es lo sucedido durante la pandemia, ya que la mayoría de las personas empezaron a trabajar desde casa debido a la contingencia, muchos **ciberdelincuentes** aprovecharon esto para revivir viejos **Malware** como crear nuevos **Malwares**, esto provocó una gran cantidad de ataques y nuevas formas de estafas, como resultado muchas personas fueron víctimas de robo de información, fraude y robo bancario, dada la necesidad de más expertos en el tema del **pentesting** de **Android** cree este curso que aborda los temas más básicos como técnicas muy avanzadas de simulaciónes de ataques reales, si te interesa ser un experto en el **penstesting** y **hacking** de **Android** te invito a seguirme en el curso. Este curso fue creado pensando en los escenarios mas reales que puedan proporcionarte los mejores ejemplos de ataques en la vida real.

Hacking Ético 2022: Curso de Metasploit Framework

Si quieres aprender como los expertos en hacking usan Metasploit como expertos te invito a seguir mi curso con más de **44.333 estudiantes satisfechos.**

CUPON DEL 80% DE DESCUENTO AQUÍ EN MI PAGINA WEB: https://bit.ly/3KMs4BX

Únete a mis **62.139** estudiantes de **Udemy** y aprende conmigo.



Descripción

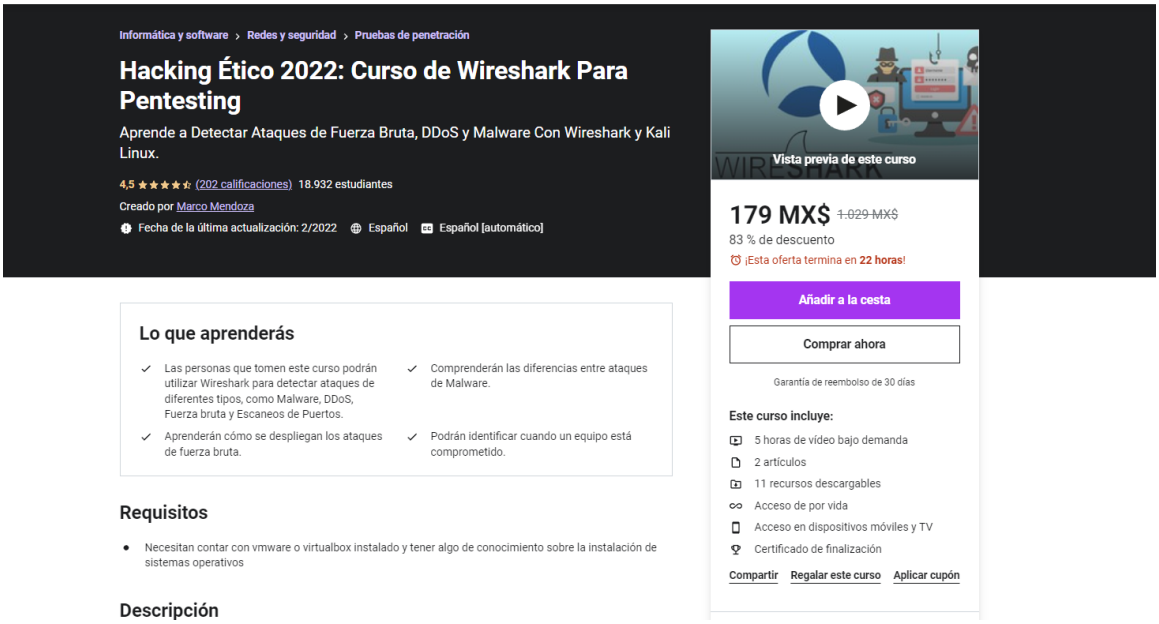
En este curso aprenderás como usar e implementar Metasploit Framework para que posteriormente puedas acoplarlo en tus pruebas de penetración y evaluación de vulnerabilidades, empezaremos con una pequeña introducción al temario del curso, después pasaremos a configurar todas las máquinas virtuales que usaremos a lo largo del curso, en este caso trabajaremos con Windows 10 y Android 11, enseguida estaremos viendo algunas herramientas que van de la mano de Metasploit Framework y daremos un vistazo rápido a la interfaz gráfica conocida como Armitage, en el siguiente capítulo veremos los módulos que conforman Metasploit Framework y también daremos un vistazo a la carpeta raíz de Metasploit para ver cómo está organizado, paso siguiente pasaremos a ver algunos auxiliares que nos ayudaran a tener una visión más amplia a la hora de buscar información de servicios que se están ejecutando bajo los protocolos más usados en redes locales, después veremos cómo desplegar ataques con Metasploit en sistemas operativos actuales, también veremos la eficiencia y deficiencia de Metasploit desplegado en diferentes entornos así como la forma de integrar nuevos exploits para personalizar nuestro Metasploit Framework, en la sección de ataques veremos como explotar la vulnerabilidad de Log4Shell utilizando drivers y aplicaciones webs vulnerables, paso siguiente troyanizaremos la App original de facebook para hackear android y tener acceso con privilegios, si quieres dominar Metasploit como un maestro te invito a seguir este curso.

Hacking Ético 2022: Curso de Wireshark Para Pentesting

Si quieres aprender cómo detectar malware, ataques DDoS, y ataques de fuerza bruta sin alterar la estructura del equipo y sin usar ingeniería inversa, este es tu curso, **únete a mis 18.932 estudiantes satisfechos.**

CUPON DEL 80% DE DESCUENTO AQUÍ EN MI PAGINA WEB: https://bit.ly/2XlvqZf

Únete a mis **62.139** estudiantes de **Udemy** y aprende conmigo.



Descripción

Hoy en día las amenazas informáticas han evolucionado de una manera muy drástica, desde Malware inteligente, hasta Ataques a gran escala que pueden provocar el cese de operaciones de una empresa, y pensando en cómo prevenir este tipo de amenazas sin alertar a los atacantes se creó este curso. Iniciaremos con una breve introducción al temario del curso, después comenzaremos con la instalación del entorno virtual que utilizaremos a lo largo de las clases, también repasaremos la importancia de evitar comprometer el sistema y daremos una introducción completa a los ataques más utilizados contra empresas, a continuación daremos un repaso de la interfaz gráfica de Wireshark, continuaremos con el análisis de paquetes para empezar a familiarizarnos con los métodos de filtrado que podemos aplicar con Wireshark, después explicaremos de forma rápida como implementar reglas de firewall y como aplicar una línea base de trabajo para tener una referencia del tráfico que circula por la red, enseguida empezaremos a ver cómo podemos crear y utilizar filtros para agilizar la búsqueda de amenazas, y por ultimo veremos cómo identificar tanto Malware como Ataques de fuerza bruta, si estás interesado en expandir tus conocimientos sobre seguridad, este es tu curso, te invito a que te suscribas para que me acompañes a lo largo de esta aventura.

Contenido

Contenido

Introducción a la Programación

Los programas de computadora son una secuencia de instrucciones para la computadora que dan detalles de lo que se necesita lograr y los detalles de los datos que se usarán. Las instrucciones a la computadora se dan usando sentencias. “Declaración” es una palabra que se usa con mucha frecuencia en el lenguaje general. Tiene un significado similar a la información, pero connota autenticación y afirmación y se espera que sea mucho más confiable. Las declaraciones se hacen en los tribunales de justicia, por políticos y en conferencias de prensa por otros. En general, una declaración tiene los siguientes atributos:

1. Cubre algún aspecto de manera integral.

2. Es preciso en sintaxis y semántica.

3. Es emitido por una persona interesada y conocedora de la situación en cuestión.

4. Está mayormente escrito y en papel.

5. Cuando se expida oralmente, se autenticará y autorizará.

6. Las declaraciones no se dan con frivolidad.

Es por eso que la palabra "Declaración" se usa con respecto a los programas de computadora. En años anteriores, una declaración se acomodaba en una línea. Una línea y una declaración eran sinónimos. En aquellos días, una línea podía acomodar solo 80 caracteres, pero surgió la necesidad de dividir la declaración en varias líneas y los programadores de computadoras idearon un método para continuar la declaración en la siguiente línea. Pero las pantallas de las computadoras modernas pueden acomodar más de 80 caracteres por línea, y algunas pantallas pueden acomodar hasta 255 caracteres por línea. Otros tienen la facilidad de desplazamiento horizontal y por lo tanto admiten más de 80 caracteres por línea. La mayoría de los lenguajes de programación modernos permiten hasta 255 caracteres por línea y también permiten múltiples líneas en una sola declaración. En la actualidad, una declaración en un programa tiene los siguientes atributos:

1. Una declaración contiene al menos una instrucción.

2. Una declaración puede contener múltiples instrucciones.

3. Contiene tanto las instrucciones como las referencias a los elementos de datos que son procesados por la instrucción.

4. Una declaración puede estar contenida en una línea o puede abarcar varias líneas. Cuando la declaración se extiende a lo largo de varias líneas, es práctica general prefijar la declaración continua con un carácter especialmente designado. El carácter de continuación difiere de un lenguaje de programación a otro. En los lenguajes de programación modernos, esto no es un requisito.

5. Generalmente, una declaración termina con un carácter especialmente designado. El carácter terminador de declaración más utilizado es el punto y coma (;).

6. Las declaraciones deben seguir la sintaxis prescrita.

Todos los lenguajes de programación especifican un conjunto de reglas de sintaxis para escribir sentencias en ese lenguaje.

Sintaxis

Cada declaración en un programa de computadora tiene que adherirse a un conjunto de reglas especificadas por el lenguaje de programación. Este conjunto de reglas se conoce comúnmente como sintaxis para ese lenguaje. La sintaxis se define como "la disposición de palabras y frases para crear oraciones bien formadas en un idioma" por Google. El Diccionario Merriam Webster define la sintaxis como "disposición de palabras en una oración, cláusulas y frases". Wikipedia define la sintaxis como “el conjunto de reglas que define la combinación de símbolos que se consideran documentos o fragmentos correctamente estructurados en ese idioma” en el contexto de la programación informática. La sintaxis para lenguajes de programación tiene las siguientes características.

1. El programa comienza con una palabra clave especialmente designada para indicar el comienzo de un programa.

2. El programa finaliza con una palabra clave especialmente designada que le dice a la computadora que la ejecución del programa ha finalizado y que se pueden realizar las acciones de cierre necesarias.

3. Todas las declaraciones del programa deben estar incrustadas entre estas palabras clave de inicio y finalización del programa.

4. Utiliza palabras y frases definidas por el lenguaje de programación. Las palabras son de dos tipos: palabras definidas por los lenguajes de programación

y palabras definidas por el programador. Las palabras definidas por el lenguaje de programación a menudo se denominan “palabras clave”.

5. Las palabras clave incluyen determinados símbolos, como operadores aritméticos (+, −, *, /, ^), paréntesis, símbolos relacionales (<, >, =) y cualquier otro símbolo específico del lenguaje de programación.

6. Las palabras clave y las palabras definidas por el programador son mutuamente excluyentes. El programador no puede utilizar las palabras clave para ningún otro propósito que no sea el definido por el lenguaje de programación. Las palabras definidas por el usuario suelen ser declaraciones de datos.

7. Generalmente, la declaración comienza con una palabra clave y es seguida por palabras definidas por el usuario y, en algunos casos, otras palabras clave. Las excepciones a esta regla se encuentran especialmente en declaraciones aritméticas, en cuyo caso la declaración puede comenzar con una palabra clave definida por el programador.

8. Existen reglas para organizar las palabras clave, los símbolos y las palabras definidas por el programador, y deben cumplirse estrictamente. No se permite desviarse de estas reglas bajo ninguna circunstancia. Por supuesto, algunos lenguajes de programación como COBOL (Common Business Oriented Language) permiten el uso de algunas palabras con fines de documentación.

9. Una declaración en la mayoría de los lenguajes de programación termina con un carácter especialmente designado. Un punto y coma (;) es el carácter de terminación de declaración más utilizado.

El incumplimiento de estas reglas de sintaxis daría lugar a lo que se conoce como "errores de sintaxis". El primer paso para convertir el código fuente en código ejecutable es comprobar si hay errores de sintaxis en el programa. El compilador comienza la conversión del código fuente a código ejecutable solo cuando no hay absolutamente ningún error de sintaxis en el código fuente del programa. Cuando el compilador del programa encuentra un error de sintaxis, detiene

todos los pasos posteriores para convertir el código fuente en un código ejecutable y genera los errores de sintaxis para que el programador pueda corregirlos y volver a enviar el programa para su compilación.

Tipos de Declaraciones

Las declaraciones se clasifican de dos maneras: por la función que desempeñan y por la complejidad de las declaraciones. Según la función que cumplen, las sentencias se clasifican en los siguientes tipos:

1. Sentencias de declaración: las sentencias de declaración se utilizan para declarar los datos que se utilizan en el programa. Las sentencias de declaración también declaran cualquier biblioteca personalizada o especial distinta de las proporcionadas por el proveedor en el SDK (Kit de desarrollo de software) utilizado por el programa.

2. Sentencias de asignación: Las sentencias de asignación asignan un valor a una variable ya declarada anteriormente en el programa. El valor a asignar puede ser otra variable, un valor constante proporcionado dentro del programa, o es el resultado de la evaluación de una expresión aritmética. Aprenderemos acerca de las expresiones en los capítulos posteriores.

3. Declaraciones de entrada/salida: estas declaraciones, como su propio nombre lo indica, reciben datos del mundo exterior o entregan datos al mundo exterior.

4. Sentencias de control: Las sentencias de control impiden que la ejecución del programa vaya a la siguiente sentencia secuencial y pueden

cambiar la ejecución a otra sentencia, que no necesita ser la siguiente sentencia en el orden secuencial. Nos ayudan a tomar decisiones programadas y procesar la información según sea necesario.

5. Bucles: Los bucles son bloques de instrucciones que se ejecutan varias veces repetidamente en función de una condición. Estas son declaraciones muy útiles en la programación.

6. Declaraciones de manejo de errores: incluso cuando tomamos sumo cuidado para evitar errores cuando el programa se está ejecutando, aún pueden aparecer errores debido a errores en los datos, resultados inesperados y el entorno. Usamos declaraciones de manejo de errores para atrapar las condiciones de error y tomar medidas correctivas para garantizar que el programa se cierre sin problemas o para alejar el programa de la condición de error a una condición normal.

7. Llamadas al sistema: por lo general, los programas se escriben utilizando las facilidades proporcionadas por el lenguaje de programación. A veces, es posible que necesitemos utilizar las funciones proporcionadas por el sistema operativo que no proporciona el lenguaje de programación. Esto se logra mediante las sentencias de llamada al sistema. Estos son más utilizados por programadores avanzados.

8. Comunicación entre procesos: estas declaraciones tampoco son las declaraciones comunes y corrientes utilizadas por todos los programadores. Estos son utilizados por programadores avanzados. Estas declaraciones se utilizan cuando dos programas que se ejecutan simultáneamente en la computadora necesitan comunicarse entre sí para intercambiar información. Estos se utilizan mucho en el desarrollo del software del sistema y el software en tiempo real.

9. Manejo de interrupciones: los programadores utilizan estas declaraciones en el desarrollo de software para el manejo de dispositivos. Cualquier dispositivo conectado a la CPU coloca una interrupción en la CPU cuando necesita comunicarse. Estas también son declaraciones avanzadas utilizadas en el desarrollo de software de sistema y software en tiempo real.

10. Manejo de dispositivos: todos los periféricos, incluidas impresoras, escáneres, otras máquinas, aviones y cohetes, necesitan un software especial. Por lo general, se los conoce como controladores de dispositivos. Los controladores de dispositivos son programas de software que manejan los dispositivos y hacen que realicen las funciones deseadas. Estos tienen llamadas y sintaxis especiales.

11. Declaraciones iniciales y finales: cada programa comienza con una declaración especial. Esto le dice a la computadora que el programa comienza a ejecutarse, y esta instrucción es seguida por una serie de instrucciones para ser ejecutadas por la computadora. De manera similar, cada programa termina con una declaración especial. Le dice a la computadora que la ejecución del programa se completó sin problemas para que pueda cerrar el programa, eliminarlo de la RAM y liberar toda la RAM asignada para los elementos de datos utilizados en el programa, así como liberar los dispositivos. utilizado por el programa. Estos son parte de cada programa.

12. Declaraciones de documentación: Una cosa es segura en el desarrollo y programación de software. Cada programa desarrollado que se pone en producción necesita ser modificado y mejorado durante su vida útil. Es posible que el programador original que lo escribió se haya ido o no esté disponible actualmente para mantener el programa. Por lo tanto, los programas deben escribirse de tal manera que otro programador los entienda y pueda mantenerlos. Los lenguajes de programación son normalmente de naturaleza críptica. Por lo tanto, todos los lenguajes de programación proporcionan una característica llamada "declaraciones de comentarios". Las declaraciones de

comentarios son completamente ignoradas por las computadoras mientras compilan y preparan el código ejecutable. Estos son para la referencia de los programadores que mantienen los programas. Por lo general, un carácter especialmente designado se antepone a la declaración para indicar que esta declaración es una declaración de comentario utilizada con el propósito de documentar la lógica del programa.

Declaraciones de Asignación

Las declaraciones de asignación forman la parte principal de cualquier programa. Es la declaración en la que se lleva a cabo el trabajo real. Las declaraciones de asignación se toman del campo de las matemáticas. En matemáticas, una declaración de asignación se ve así:

$$a \leftarrow b + c + d - e$$

La especificación de esta expresión es:

1. Evalúa la expresión de (b + c + d − e).

2. Asigne el valor resultante a "a".

3. El símbolo de la flecha es el símbolo de la asignación. La punta de flecha indica la dirección de la asignación.

4. Generalmente, la punta de la flecha apunta al lado izquierdo de la expresión.

5. Solo se permite una variable en el lado izquierdo de la flecha.

Las mismas reglas se aplican a las declaraciones de asignación en la programación, pero el símbolo de asignación no es el signo de flecha. En la mayoría de los lenguajes de programación, es el signo igual a (=). Una declaración de asignación en un programa de computadora se ve así:

$$A = B + C$$

La especificación de esta expresión es:

1. Tome el contenido de la ubicación de memoria B

2. Tome el contenido de la ubicación de memoria C

3. Añadir B y C

4. Asigne el resultado de la suma a la ubicación de memoria A

Una desviación que tienen las asignaciones en la programación sobre las declaraciones de asignación matemática es que la variable en el lado izquierdo del símbolo de asignación también puede estar allí en el lado derecho del símbolo de asignación. Por ejemplo, "A = A + B" está permitido en los programas de computadora, pero no en las tareas matemáticas. Esta afirmación da como resultado:

1. Tome el contenido de la ubicación de memoria A

2. Tome el contenido de la ubicación de memoria B

3. Suma A y B

4. Asigne el resultado a la ubicación de memoria A

De hecho, este tipo de declaración es muy común en la programación. Generalizando las declaraciones de asignación, podemos enumerar las siguientes reglas:

1. Las declaraciones de asignación contendrían el símbolo de asignación, que aparece solo una vez, que generalmente es el signo igual a (=).

2. En el lado izquierdo de la declaración de asignación debe haber una sola variable.

3. En el lado derecho del símbolo de asignación, puede haber:

A. Una constante, que puede ser un número o un literal (una cadena de caracteres entre comillas).

B. Una expresión matemática que contiene todas las variables, todas las constantes, una combinación de variables y constantes, o una combinación de expresiones.

C. La variable del lado izquierdo de la declaración de asignación puede ser parte de la expresión del lado derecho de la declaración de asignación.

4. Durante la ejecución del programa, se evalúa la expresión del lado derecho del símbolo de asignación y el resultado se asignaría y almacenaría en la ubicación de memoria simbolizada por la variable del lado izquierdo del símbolo de asignación.

Expresiones Aritméticas, Relacionales y Lógicas

Introducción a las expresiones

Casi todas las sentencias contienen al menos una expresión. Wikipedia define una expresión así: “una expresión en programación informática es una combinación de valores explícitos, constantes, variables, operadores y funciones, que se interpretan de acuerdo con las reglas particulares de precedencia y asociación para el lenguaje de programación específico”. Enumeré los atributos de las expresiones en el contexto de la programación informática para comprenderlas mejor:

1. Una expresión puede constar de al menos dos variables, una constante o una combinación de variables y constantes.

2. Las variables y constantes en la expresión se combinan usando símbolos matemáticos, relacionales o lógicos, generalmente denominados "operadores".

3. La expresión es susceptible de evaluación utilizando reglas aritméticas, relacionales o lógicas.

4. La evaluación de una expresión en un enunciado arroja solo un valor que puede usarse para la asignación a una variable o en la toma de decisiones programada. Es decir, las expresiones se pueden usar en sentencias de asignación y sentencias de control. Las expresiones no se utilizan en un modo independiente.

5. Las expresiones, cuando se usan en declaraciones de asignación, deben estar en el lado derecho del símbolo de asignación. Una expresión nunca puede estar en el lado izquierdo de un símbolo de asignación.

Tipos de Expresiones

Las expresiones utilizadas en la programación informática son de tres tipos:

1. Expresiones aritméticas

2. Expresiones relacionales

3. Expresiones lógicas

Analicemos cada uno de estos tipos con mayor detalle a continuación.

Expresiones Aritméticas

Las expresiones aritméticas se utilizan para resolver ecuaciones aritméticas y calcular los valores requeridos. Estas expresiones pueden incluir tanto variables como constantes. Las variables y constantes utilizadas en las expresiones

aritméticas deben ser de tipo numérico. Por supuesto, algunos lenguajes de programación permiten el uso de variables y constantes de tipo carácter en expresiones aritméticas. Pero, solo se pueden usar para sumar. Cuando se utilizan dos cadenas de caracteres más en una expresión aritmética mediante un operador de suma, se concatenarán entre sí. Las variables y constantes en las expresiones aritméticas se unen para formar una expresión mediante los operadores aritméticos. Los siguientes son los operadores aritméticos:

1. Símbolo de suma +

2. Símbolo de resta − (un guión)

3. Símbolo de multiplicación * (un asterisco)

4. Símbolo de división / (una barra oblicua)

5. Símbolo de exponenciación ^ (un signo de intercalación)

6. Abrir paréntesis “(”

7. Cerrar paréntesis “)”

8. Para la raíz cuadrada, no se asigna ningún símbolo. Por lo general, se logra mediante una rutina de biblioteca. La rutina de biblioteca específica utilizada para encontrar la raíz cuadrada difiere entre los lenguajes de programación.

En las matemáticas de la vida real, usamos llaves {&} y corchetes [&] junto con paréntesis, "(" & ")", cuando se necesita más de un conjunto de paréntesis en la expresión matemática. En la programación, usamos solo las llaves curvas en expresiones aritméticas. Aquí hay algunos ejemplos válidos de expresiones aritméticas:

- a + b
- a + b − c
- a + b * c
- a + (b * c)
- (a + b) * c
- a − b * c / d
- (a + b) ^ 2
- (a + b) ^ 2 − (a − b) ^ 2 / d
- ((a + b) ^ 2 − (a − b) ^ 2) / d

Precedencia en las Reglas de Evaluación

Al evaluar expresiones aritméticas, las computadoras suelen seguir estas reglas de precedencia:

1. La evaluación procede de izquierda a derecha cuando el nivel de precedencia de los operadores es el mismo.

2. El símbolo de suma y los símbolos de resta tienen la prioridad más baja en la precedencia de evaluación. Ambos tienen la misma precedencia.

3. Los símbolos de multiplicación y división tienen la misma precedencia. Tienen mayor precedencia que los símbolos de suma y resta.

4. El símbolo de exponenciación tiene mayor precedencia sobre los símbolos de multiplicación y división.

5. Los paréntesis tienen mayor precedencia que el resto de los operadores aritméticos.

Ahora, usando estas reglas, veamos cómo la computadora evalúa las expresiones de ejemplo anteriores:

1. a + b: solo hay un operador. La computadora simplemente suma los valores de ambas variables y entrega el resultado.

2. a + b – c: aquí tenemos dos operadores con la misma precedencia. Por lo tanto, la computadora sumará el valor de **a** con el valor de **b** y luego restará el valor de **c** de la suma y entregará el resultado.

3. a + b * c: aquí tenemos dos operadores con diferentes niveles de precedencia. Por lo tanto, la computadora primero evaluará la parte de la expresión unida por el operador con mayor precedencia. Entonces, multiplicará el valor de **b** con el valor de **c**. Luego agregará el resultado al valor de **a** y luego entregará el resultado.

4. a + (b – c): aquí tenemos tres operadores con diferentes niveles de precedencia. La computadora evaluará la expresión entre paréntesis. Entonces, el valor de **c** se restaría del valor de **b** y luego el resultado se sumaría al valor de **a**. Entonces el resultado sería entregado.

5. (a – b) * c: aquí tenemos tres operadores con diferentes niveles de precedencia. La computadora evaluará la expresión entre paréntesis. Entonces, el valor de **b** se restaría del valor de **a** y luego el resultado se multiplicaría por el valor de **c**. Entonces el resultado sería entregado.

6. a – b * c / d: aquí tenemos múltiples operadores con diferentes niveles de precedencia.

A. Siguiendo la regla #1 de evaluación, la evaluación procede de izquierda a derecha. Por lo tanto, **b** se multiplicaría por **c**.

B. El resultado se dividiría entonces por **d**.

C. Entonces el resultado se restaría del valor de **a**.

D. Entonces el resultado sería entregado.

7. (a + b) ^ 2: la evaluación en esta expresión es bastante sencilla. La expresión entre paréntesis se evaluaría primero. Luego se elevaría por el valor del exponente, **2**, y luego se entregaría el resultado.

8. (a + b) ^ 2 – (a – b) ^ 2 / d: aquí tenemos múltiples operadores con diferentes niveles de precedencia.

A. Primero, la computadora evaluaría las expresiones entre paréntesis.

B. Luego dividiría el resultado de la expresión **(a – b) ^ 2** por **d**.

C. Luego el resultado se sumaría al resultado de la expresión **(a + b) ^ 2**.

D. El resultado sería entregado.

9. ((a + b) ^ 2 – (a – b) ^ 2) / d: aquí tenemos múltiples operadores con diferentes niveles de precedencia.

A. Primero, la computadora evaluaría las expresiones en los paréntesis más internos. Entonces se evaluaría el paréntesis exterior.

B. Luego, el resultado de la expresión **(a – b) ^ 2** se restaría del resultado de la expresión **(a + b) ^ 2**.

C. Luego dividirá el resultado por **d**.

D. Entonces el resultado sería entregado.

Como puede ver, las expresiones en las viñetas #8 y #9 son similares excepto por la ubicación de los paréntesis, que cambia la forma en que se evalúa la expresión. Ambas expresiones darían resultados diferentes.

Por lo tanto, debemos tener cuidado al programar las expresiones aritméticas, especialmente al usar los operadores con diferentes niveles de precedencia. Como precaución, se recomienda utilizar los operadores de paréntesis generosamente para evitar confusiones acerca de los valores de precedencia para obtener el resultado deseado y preciso. También se recomienda evitar escribir expresiones aritméticas largas, ya que puede resultar muy confuso al depurar el programa. Es mejor dividir las expresiones largas en múltiples expresiones más pequeñas y escribirlas en diferentes líneas.

Al usar expresiones aritméticas en enunciados aritméticos, se sugieren las siguientes precauciones para obtener un resultado sin errores.

Expresiones Relacionales

Las expresiones relacionales relacionan una expresión con otra y determinan si la relación entre las dos es verdadera o falsa. Las expresiones relacionales se

utilizan para la toma de decisiones y se utilizan en declaraciones de control. No se utilizan en sentencias de asignación. Las expresiones en las expresiones relacionales se unen mediante los operadores relacionales. Los operadores relacionales especifican el tipo de relación a establecer entre ambas expresiones.

Si bien los paréntesis no son operadores relacionales, a menudo se usan para encerrar una expresión relacional. Es una buena práctica de programación encerrar toda la expresión relacional en un conjunto de paréntesis. Pero se pueden ver variaciones en algunos lenguajes de programación. Por ejemplo, **COBOL** usa texto completo, como **MENOS QUE, MAYOR QUE, NO IGUAL**, etc. En algunos lenguajes de programación se utilizan los caracteres **le, ge, eq, etc**. Debe consultar el manual del lenguaje de programación o las páginas de ayuda para saber exactamente qué operadores relacionales se usan en ese lenguaje.

Como se señaló anteriormente en este capítulo, una expresión puede contener una variable, una constante o una combinación de variables y constantes. Necesitamos seguir las siguientes reglas al escribir expresiones relacionales:

1. Las expresiones deben estar en diferentes lados de un operador relacional. El operador relacional estaría en el medio con una expresión en su lado izquierdo y la otra expresión estaría en su lado derecho.

2. Una expresión relacional puede contener solo un operador relacional. No podemos tener más de un operador relacional en una expresión relacional.

3. Ambas expresiones incluidas en la expresión relacional deben ser del mismo tipo de datos. Si la expresión del lado izquierdo es de tipo numérico, entonces la del lado derecho debe ser de tipo numérico. Si una de las expresiones es de tipo numérico, la otra no puede ser de otro tipo que no sea numérico.

4. Si ambas expresiones son de tipo numérico, su precisión podrá, sin embargo, ser diferente. Si uno es de tipo entero, el otro puede ser de precisión simple o doble. El resultado, sin embargo, dependería del valor real y del operador relacional.

5. Algunos lenguajes de programación permiten diferentes tipos de datos de expresiones. Uno puede ser de tipo numérico y el otro puede ser de tipo carácter. Pero los resultados serían impredecibles. Es decir, el lenguaje de programación no se responsabiliza de encontrar errores de programación lógica. Corresponde al programador escribir programas libres de defectos. Incluso si el lenguaje de programación lo permite, no es correcto escribir expresiones relacionales usando dos expresiones de diferentes tipos de datos a cada lado del operador relacional.

6. El resultado que arroja la evaluación de una expresión relacional es VERDADERO (uno o SÍ) o FALSO (cero o NO).

Operador	Función
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual que
!=	distinto que

Necesitamos tomar nota de un aspecto importante aquí. Si bien está permitido usar constantes en expresiones relacionales, no tiene sentido usar solo constantes en ambos lados del operador relacional. ¿De qué sirve comparar 3 y 4 para ver si 4 es mayor que 3? Todos los operadores relacionales tienen menor precedencia que todos los operadores aritméticos. Entonces, si una expresión aritmética se incluye en una expresión relacional, se evaluaría primero antes de evaluar la expresión relacional. Ahora consideremos algunos ejemplos de expresiones relacionales:

1. x > y: esta expresión compara el valor de **x** con el valor de **y**. Entonces, si el valor de **x** es mayor que **y**, devolverá el resultado de **VERDADERO**. Si el valor de **y** es igual o mayor que el valor de **x**, devolverá un resultado **FALSO**.

2. a + b > c: en esta expresión, sumará el valor de **a** al valor de **b** y solo entonces comparará el valor resultante con el valor de **c**. Finalmente, devolverá un resultado de **VERDADERO** solo cuando **a + b** sea mayor que el valor de **c**.

3. (a + b) > c: esta expresión es similar a la expresión en la viñeta #2 excepto que se usa un paréntesis para hacer que el proceso de evaluación sea más explícito.

4. (a + b) < (c + d): esto tiene expresiones aritméticas en ambos lados del operador relacional y ambos están entre paréntesis. Ambas expresiones se evaluarían primero y luego se compararían los resultados. Entonces se entregaría el resultado apropiado, ya sea **VERDADERO** o **FALSO**.

5. ((a + b)/c) >= (x + y/z): este es otro ejemplo de comparación de dos expresiones aritméticas. En la expresión del lado izquierdo del operador relacional, el valor de **a** se sumaría al valor de **b** y luego la suma se dividiría por **c**. En la otra expresión, el valor de **y** se dividiría por el valor de **z** y luego el cociente se sumaría al valor de **x**. Luego, los resultados de ambas expresiones se compararían para obtener el resultado de la expresión relacional.

Ahora, necesitamos entender la diferencia entre algunos símbolos confusos. Por supuesto, el significado de los símbolos no es realmente confuso, pero si no tenemos cuidado, podemos usarlos incorrectamente. Déjanos considerar algunos ejemplos. Supongamos que el valor de **a** es **4**; el valor de **b** es **4**; el valor de **c** es **4.001**. Ahora usemos estos valores en nuestros ejemplos:

1. (a == b): esto se evaluaría como **VERDADERO**, ya que el valor de **a** y **b** es **4**.

2. (a != b): esta expresión se evalúa como **FALSO**, ya que los valores de **a** y **b** son iguales.

3. (a <= b): esta expresión se evaluaría como **VERDADERA**. b no es menor que **a**, pero es igual a **a**.

4. (a >= b): esta expresión también se evaluaría como **VERDADERO**. **a** no es mayor que **b**, pero es igual a **b**.

5. (a == c): esto se evalúa como **FALSO** porque, si bien la diferencia entre los valores de **a** y **c** es muy pequeña, sigue siendo significativa. Pero **a** y **c** no son iguales, por pequeña que sea la diferencia. Los seres humanos podemos considerar que la diferencia es insignificante a todos los efectos prácticos, pero una computadora consideraría significativa cualquier diferencia, independientemente de su magnitud.

6. (a != c): esta expresión se evalúa como **VERDADERA**, ya que los valores de **a** y **c** no son iguales.

7. (a >= c): esta expresión se evalúa como **FALSO**. El valor de **c** es ciertamente mayor que el de **a** a la vista de la computadora.

8. (a <= c): esta expresión se evalúa como **VERDADERA** ya que a es ciertamente menor que el valor de **c**

Expresiones Lógicas

Las computadoras se construyen utilizando chips de circuitos integrados (IC), que se han transformado en chips integrados a gran escala (chips LSI) y ahora en chips de circuitos integrados a muy gran escala (chips VLSI). Estos chips se construyen sobre la base de circuitos lógicos denominados puertas, es decir, la puerta AND, la puerta OR y la puerta NOT. Estas puertas se construyen utilizando diodos y transistores. Si bien una explicación detallada de estos componentes de hardware está fuera del alcance de este libro, es necesario que entendamos los conceptos básicos de estos chips, para que tengamos una comprensión más clara de las expresiones lógicas que usamos en la programación de computadoras.

Estas puertas pueden tener múltiples entradas, pero solo una salida. La combinación de entradas y la salida correspondiente se representa en una tabla denominada "tabla de verdad" del chip.

Ahora la explicación es la siguiente. La puerta AND produce una salida solo cuando todas las entradas están presentes. En otras palabras, la puerta AND produce una salida VERDADERA solo cuando todas las entradas son VERDADERAS. La puerta OR produce una salida VERDADERA cuando cualquiera de las entradas es VERDADERA. La puerta OR produce una salida FALSA solo cuando todas las entradas son FALSAS. La puerta NOT produce una salida opuesta a la entrada. Es decir, si la entrada es VERDADERA, la salida sería FALSA, y si la entrada es FALSA, la salida sería VERDADERA.

En las puertas lógicas, hay otras puertas construidas usando estas puertas básicas. Son puertas NAND, NOR y XOR. Una puerta NAND es una combinación de una puerta AND y una puerta NOT. Es decir, la salida de una puerta AND se alimenta como entrada a una puerta NOT. Una puerta NOR es aquella en la que la salida de una puerta OR se alimenta como entrada a una

puerta NOT. Una puerta XOR (o puerta OR exclusiva) es una variedad especial de puerta OR.

La puerta XOR es como una puerta OR en sus combinaciones de entrada/salida, excepto que cuando todas las entradas son VERDADERAS, devuelve una salida FALSA. Es decir, una puerta XOR devuelve una salida VERDADERA solo cuando cualquiera de sus entradas es VERDADERA, pero no cuando todas las entradas son VERDADERAS.

Estas puertas lógicas son los componentes básicos de nuestras computadoras electrónicas y también son la base de nuestras expresiones lógicas.

Ahora, volviendo a nuestra discusión sobre la programación, las entradas son expresiones relacionales y la salida es el resultado de la evaluación de la expresión lógica. Una expresión lógica es una combinación de expresiones unidas por operadores lógicos. A diferencia de las expresiones relacionales, podemos tener más de un operador lógico en una expresión lógica. Los operadores lógicos son AND (&&), OR (||), NOT (!) y XOR. XOR rara vez se usa en la fraternidad de programación y no muchos lenguajes de programación lo admiten.

Todos los operadores lógicos tienen el mismo nivel de precedencia, por lo que las expresiones lógicas siempre se evalúan de izquierda a derecha. Las expresiones colocadas a ambos lados del operador lógico pueden ser expresiones relacionales o variables de tipo booleano. Supongamos que **a**, **b** y **c** son expresiones válidas para usar en expresiones lógicas. Ahora consideremos algunos ejemplos:

1. a && b: Ahora la salida sería la siguiente:

A. Cuando ambas expresiones se evalúan como VERDADERO, el resultado entregado sería VERDADERO.

B. En todos los demás casos, la salida entregada sería FALSO.

2. a || b: Ahora la salida sería la siguiente:

A. Cuando ambas expresiones se evalúan como FALSO, el resultado entregado sería FALSO.

B. En todos los demás casos, la salida entregada sería VERDADERO.

3. a && b || c: esta expresión evalúa primero (a && b), y el resultado se usaría para relacionarse lógicamente con **c**. Los resultados serían los siguientes:

A. Cuando **a**, **b** y **c** se evalúan como VERDADERO, el resultado sería VERDADERO.

B. Cuando **a** se evalúa como FALSO, pero **b** y **c** se evalúan como VERDADERO, el resultado seguirá siendo VERDADERO, ya que **c** está conectado a la otra parte de la expresión mediante el operador OR.

C. Cuando **a** se evalúa como FALSO y **b** también se evalúa como FALSO pero **c** se evalúa como VERDADERO, la salida seguirá siendo VERDADERA porque una expresión del operador OR es VERDADERO.

D. Cuando **a** es VERDADERO pero **b** es FALSO y **c** es VERDADERO, la salida seguirá siendo VERDADERA porque una expresión del operador OR es VERDADERO.

B. Cuando las tres expresiones, **a**, **b** y **c**, se evalúan como FALSO, el resultado entregado sería FALSO.

4. !a: este es un ejemplo del uso del operador NOT. El operador NOT devuelve un valor opuesto al valor entregado por la evaluación de la expresión. La expresión se evalúa de la siguiente manera:

A. Si la expresión **a** se evalúa como VERDADERO, el resultado entregado sería FALSO.

B. Si la expresión a se evalúa como FALSO, el resultado entregado sería VERDADERO.

Mejores Prácticas en el Uso de Expresiones Lógicas

Si bien no es realmente necesario usar paréntesis en expresiones lógicas, es mejor encerrar una expresión lógica en un conjunto de paréntesis. Las expresiones lógicas se usarían invariablemente en declaraciones de control y los paréntesis establecerían la expresión lógica como una entidad separada en la declaración. Esto ayuda a comprender la lógica del programa, especialmente durante la depuración y el mantenimiento del programa.

Una expresión lógica contendría invariablemente un mínimo de dos expresiones relacionales. Siempre es una buena práctica encerrar cada expresión relacional en un conjunto de paréntesis para que las expresiones evaluadas por el operador lógico se destaquen claramente entre sí. Esto facilita la comprensión de la lógica del programa y el mantenimiento del programa.

Si bien podemos tener cualquier cantidad de operadores lógicos en una expresión lógica, es mejor restringir la cantidad de operadores lógicos en una expresión lógica a un máximo de tres. Si usamos más de tres operadores lógicos en una expresión lógica, sería bastante difícil de descifrar al depurar el programa o durante el mantenimiento del programa.