C ++ hace que el polimorfismo sea fácil de programar. Sin duda es posible programar el polimorfismo en lenguajes no orientados a objetos como C, pero hacerlo requiere manipulaciones complejas y potencialmente peligrosas. Esta sección trata sobre cómo C ++ puede implementar polimorfismo, funciones virtuales y vinculación dinámica internamente. Esto le dará una comprensión sólida de cómo estas capacidades realmente funcionan. Lo que es más importante, lo ayudará a apreciar la sobrecarga de polimorfismo, en términos de consumo de memoria adicional y tiempo de procesador. Esto puede ayudarlo a determinar cuándo usar el polimorfismo y cuándo evitarlo. Las clases de biblioteca estándar de C ++, como matriz y vector, se implementan sin polimorfismo y funciones virtuales para evitar la sobrecarga de tiempo de ejecución asociada y lograr un rendimiento óptimo.  
Primero, explicaremos las estructuras de datos que el compilador construye en tiempo de compilación para soportar el polimorfismo en el momento de la ejecución. Verá que el polimorfismo se logra a través de tres niveles de punteros, es decir, triple indirección. Luego, mostraremos cómo un programa en ejecución utiliza estas estructuras de datos para ejecutar funciones virtuales y lograr la vinculación dinámica asociada con el polimorfismo. Nuestra discusión explica una posible implementación; este no es un requisito de idioma  
Cuando C ++ compila una clase que tiene una o más funciones virtuales, construye una tabla de funciones virtuales (vtable) para esa clase. El vtable contiene punteros a la clase virtual  
funciones. Así como el nombre de una matriz incorporada contiene la dirección en la memoria del primer elemento de la matriz, un puntero a una función contiene la dirección de inicio en la memoria del código que realiza la tarea de la función. Un programa en ejecución utiliza el vtable para seleccionar la implementación de la función adecuada cada vez que se llama a una función virtual de esa clase. La columna más a la izquierda de la figura 12.18 ilustra los cuadros de las clases Employee, SalariedEmployee, CommissionEmployee y BasePlusCommissionEmployee.

Clase de empleados vtable  
En el vtable de la clase Employee, el primer puntero a la función se establece en 0 (es decir, nullptr), porque la función de ganancias es una función virtual pura y, por lo tanto, carece de una implementación. El puntero de la segunda función apunta a la función imprimir, que muestra el nombre completo y el número de seguro social del empleado. [Nota: hemos abreviado el resultado de cada función de impresión en esta figura para ahorrar espacio.] Cualquier clase que tenga uno o más punteros nulos en su vtable es una clase abstracta. Las clases sin punteros vtable nulos (como SalariedEmployee, CommissionEmployee y BasePlusCommissionEmployee) son clases concretas.

SalariedEmployee Class vtable  
El empleado asalariado de clase anula las ganancias de función para devolver el salario semanal del empleado, por lo que el puntero de función apunta a la función de ingresos de la clase SalariedEmployee. SalariedEmployee también anula la impresión, por lo que el puntero de la función correspondiente apunta a la función de miembro SalariedEmployee que imprime "empleado asalariado:" seguido del nombre del empleado, el número de la seguridad social y el salario semanal

ComisiónEmployee Class vtable  
El puntero de la función de ganancias en el vtable para la clase CommissionEmployee señala la función de ingresos de CommissionEmployee que devuelve las ventas brutas del empleado multiplicadas por la tasa de comisión. El puntero de la función de impresión apunta a la versión CommissionEmployee de la función, que imprime el tipo, el nombre, el número de seguro social, la tasa de comisión y las ventas brutas del empleado. Al igual que en la clase Empleado por hora, ambas funciones anulan las funciones en la clase Empleado.

BasePlusCommissionEmployee Class vtable  
El puntero de la función de ingresos en el vtable para la clase BasePlusCommissionEmployee apunta a la función de ganancias de BasePlusCommissionEmployee, que devuelve el salario base del empleado más las ventas brutas multiplicadas por la tasa de comisión. El puntero de la función de impresión apunta a la versión BasePlusCommissionEmployee de la función, que imprime el salario base del empleado más el tipo, nombre, número de seguro social, tasa de comisión y ventas brutas. Ambas funciones anulan las funciones en la clase CommissionEmployee.

Heredar funciones virtuales concretas  
En nuestro estudio de caso de Empleado, cada clase concreta proporciona su propia implementación para las funciones virtuales, las ganancias y la impresión. Has aprendido que cada clase que hereda directamente de la clase base abstracta Empleado debe implementar las ganancias para que sea una clase concreta, porque las ganancias son una función virtual pura. Sin embargo, estas clases no necesitan implementar la función de impresión, para ser consideradas concretas: la impresión no es una función virtual pura y las clases derivadas pueden heredar la implementación de print del empleado de la clase. Además, la clase BasePlusCommissionEmployee no tiene que implementar la función print o los ingresos: ambas implementaciones de funciones se pueden heredar de la clase concreta CommissionEmployee. Si una clase en nuestra jerarquía heredara las implementaciones de funciones de esta manera, los punteros vtable para estas funciones simplemente señalarían la implementación de la función que se heredaba. Por ejemplo, si BasePlusCommissionEmployee no anuló los ingresos, el puntero de la función de ingresos en el vtable para la clase BasePlusCommissionEmployee apuntaría a la misma función de ingresos que el vtable para la clase a la que CommissionEmployee apunta.

Tres niveles de punteros para implementar el polimorfismo  
El polimorfismo se logra a través de una elegante estructura de datos que involucra tres niveles de indicadores. Hemos discutido un nivel: los punteros de función en el vtable. Estos apuntan a las funciones reales que se ejecutan cuando se invoca una función virtual.  
Ahora consideramos el segundo nivel de punteros. Cada vez que se crea una instancia de un objeto de una clase con una o más funciones virtuales, el compilador adjunta al objeto un puntero al vtable para esa clase. Este puntero normalmente se encuentra en la parte frontal del objeto, pero no es necesario implementarlo de esa manera. En la figura 12.18, estos punteros están asociados con los objetos creados en la figura 12.17 (un objeto para cada uno de los tipos SalariedEmployee, CommissionEmployee y BasePlusCommissionEmployee). El diagrama muestra cada uno de los valores de miembro de datos del objeto. Por ejemplo, el objeto asalariado empleado contiene un puntero a SalariedEmployee vtable; el objeto también contiene los valores John Smith, 111-11-1111 y $ 800.00.  
El tercer nivel de punteros simplemente contiene los identificadores de los objetos que reciben las llamadas a funciones virtuales. Los controladores en este nivel también pueden ser referencias. La figura 12.18 muestra los empleados de vectores que contienen punteros de empleado.  
Ahora veamos cómo se ejecuta una llamada a función virtual típica. Considere la llamada baseClassPtr-> print () en la función virtualViaPointer (línea 69 de la Fig. 12.17).  
Suponga que baseClassPtr contiene empleados [1] (es decir, la dirección del objeto comisiónEmpleado en empleados). Cuando el compilador compila esta declaración, determina que la llamada se está realizando a través de un puntero de clase base y que la impresión es una función virtual.  
El compilador determina que print es la segunda entrada en cada uno de los vtables. Para ubicar esta entrada, el compilador observa que necesitará omitir la primera entrada. Por lo tanto, el compilador compila un desplazamiento o desplazamiento en la tabla de punteros de código objeto de máquina para encontrar el código que ejecutará la llamada de función virtual. El tamaño en bytes del desplazamiento depende del número de bytes utilizados para representar un puntero de función en una plataforma individual. Por ejemplo, en una plataforma de 32 bits, un puntero típicamente se almacena en cuatro bytes, mientras que en una plataforma de 64 bits, un puntero típicamente se almacena en ocho bytes. Suponemos cuatro bytes para esta discusión.  
El compilador genera código que realiza las siguientes operaciones [Nota: los números en la lista corresponden a los números encerrados en la figura 12.18]:  
1. Seleccione la i-ésima entrada de empleados (en este caso, la dirección del objeto commissionEmployee), y páselo como argumento para que funcione virtualViaPointer. Esto establece el parámetro baseClassPtr para que apunte a commissionEmployee.  
2. Desconéctese ese puntero para llegar al objeto commissionEmployee, que, como recordará, comienza con un puntero al CommissionEmployee vtable.  
3. Desreferencia comisiónEmpleo vtable puntero para llegar a CommissionEmployee vtable.  
4. Omita el desplazamiento de cuatro bytes para seleccionar el puntero de la función de impresión.  
5. Desprecie el puntero de la función de impresión para formar el "nombre" de la función real a ejecutar, y use el operador de llamada a función () para ejecutar la función de impresión adecuada, que en este caso imprime el tipo, nombre y número de seguro social del empleado. las ventas brutas y la tasa de comisión.