Con las funciones virtuales y el polimorfismo es posible diseñar e implementar sistemas que con extensibles con mayor facilidad. Los programas se pueden escribir para que procesen objetos en forma genérica, como objetos de clase base de todas las clases existentes en una jerarquía. Las clases que no existen durante el desarrollo del programa se pueden agregar con poca o sin modificaciones a la parte genérica del programa, siempre y cuando esas clases sean parte de la jerarquía que está siendo procesada en forma genérica. Las únicas partes del programa que necesitarán modificaciones son aquellas que requieren un conocimiento directo de la clase particular que se está agregando a la jerarquía.

Funciones virtual

Si tenemos un conjunto de clases con las formas: Circle, Triangle, Rectangle, Square, que están derivadas de la clase base
Shape
. En la programación orientada a objetos a cada una de estas figuras se le podría dar la habilidad de dibujarse a sí misma. Aunque cada clase tiene su propia función
draw
, esta es bastante diferente en cada forma. Cuando se dibuja una forma sin importar el tipo, sería agradable tratar a todas esta formas genéricamente como objetos de la clase base
Shape
.

Luego, para dibujar cualquier forma podríamos llamar a la función draw de la clase base Shape y dejar que el programa determine dinámicamente (en tiempo de ejecución) cual función
draw
de la clase derivada se debe utilizar.

Para permitir este tipo de comportamiento declaramos a draw de la clase base como función
virtual
, y sobreponemos a
draw
en cada una de las clases derivadas para dibujar la forma adecuada.

Una función de este tipo se declara precediendo la palabra clave virtual, al prototipo de la función, en la definición de la clase, por ejemplo:

```
virtual void draw ();
```

Clases base abstractas y clases concretas.

Hay casos en los que es útil definir clases, para las cuales el programador nunca pretende

instanciar ningún objeto. Tales clases se denomina clases abstractas. Debido a que estas se utilizan como clases base en situaciones de herencia, normalmente nos referimos a ellas como clases base abstractas. No es posible instanciar un objeto de una clase base abstracta.

El único propósito de una clase abstracta es proporcionar una clase base adecuada a partir de la cual las clases puedan heredar interfaces y/o implementaciones. Las clases de las que se pueden instanciar objetos se denomina clases concretas.

Por ejemplo, se puede tener una clase base abstracta TwoDimwensionalShape y derivar clases concretas tales como
Cube
,
Sphere
,
Cylinder
. Las clases base abstractas son demasiado genéricas como para definir objetos reales, y necesitamos ser mucho más específicos antes de que podamos pensar en instanciar objetos.

Para que una clase sea abstracta se debe declarar como "pura" una o más de sus funciones virtual. Una función virtual pura es aquella que tiene un inicializador = 0 en su declaración, por ejemplo:

```
virtual float earningsO   const =                0                ;
```

Polimorfismo

C++ permite el polimorfismo, que es la habilidad de los objetos de diferentes clases que están relacionados mediante la herencia para responder en forma diferente al mismo mensaje (es decir, a la llamada de función miembro). El mismo mensaje que se envía a muchos tipos de objetos diferentes toma "muchas formas", y de ahí viene el término polimorfismo.

Por ejemplo, si la clase Rectangle se deriva de la clase Quadrilateral, un objeto Rectangle es una versión más específica de un objeto
Quadrilateral
. Una operación (como el cálculo del perímetro o el área) que puede realizarse en un objeto
Quadrilateral
también puede realizarse en un objeto
Rectangle
.

El polimorfismo se implementa por medio de funciones virtual.

Cuando se hace una petición por medio de un apuntador de clase base (o referencia) , para

utilizar una función virtual, C++ elige la función sobrepuesta correcta en la clase derivada adecuada que está asociada con ese objeto. Hay muchas veces en que una función miembro no virtual está definida en la clase base y sobrepuesta en una clase derivada. Si a una función de estas se le llama mediante un apuntador de clase base al objeto de la clase derivada, se utiliza la versión de la clase base. Si la función miembro se llama mediante un apuntador de la clase derivada, se utiliza la versión de dicha clase derivada. Este comportamiento no es polimórfico.

Mediante el uso de funciones virtual y el polimorfismo, una llamada de función miembro puede causar que sucedan diferentes acciones, dependiendo del tipo de objeto que recibe la llamada. Esto le da una capacidad expresiva tremenda al programador.

El polimorfismo promueve la extensibilidad: el software que está escrito para llamar al comportamiento polimórfico se escribe en forma independiente de los tipos de objetos a los cuales se envían los mensajes. Por lo tanto los nuevos tipos de objetos que pueden responder a los mensajes existentes se pueden agregar a un sistemas, sin modificar el sistema base.

Un ejemplo concreto

El enunciado del programa sería el siguiente:

Definir una clase Shape que sea una clase base abstracta que contenga la interfaz hacia la jerarquía. Derive a TwoDimensionalShape y ThreeDimensionalShape de la clase Shape, que también serán abstractas. Utilice una función print virtual para enviar a la salida el tipo y dimensiones de cada figura. También incluye funciones virtual area y volume para que estos cálculos puedan realizarse para los objetos de cada clase concreta de la jerarquía.
Escriba un programa controlador que pruebe la jerarquía de la clase Shape.

Y la solución...

```
#include <iostream.h>
#include <math.h>
#include <conio.h>

// clase Shape

class Shape {
    public:
```

```
                    virtual double area     () const { return       0.0                     ; }
                    virtual double volume   () const { return       0.0                     ; }

                    // funcion virtual pura sobrepuesta en las clases derivadas
                    virtual void            print() const=          0                       ;
};

////////////////////////////////
// clase TwoDimensionalShape ///
////////////////////////////////

class               TwoDimensionalShape : public                    Shape                   {
    public:
                    virtual void            print () const=         0                       ;
};

// clase triangulo
class               triangulo           : public                    TwoDimensionalShape {
                    double lado1        ,                           lado2                   ,
    public:
                    triangulo           (                           double                  =
                    void fijar_triangulo (                          double                  ,
                    virtual double area  () const;
                    virtual void        print() const;
};

triangulo           ::                  triangulo                   (                       double
                    fijar_triangulo     (                           l1                      ,
}

void triangulo      ::                  fijar_triangulo             (                       double
                    lado1               =                           l1                      >
                    lado2               =                           l2                      >
                    lado3               =                           l3                      >
}

double triangulo    ::                  area                        () const {
                    double s            ;
                    s                   =(                          lado1                   +

    return          sqrt                (                           s                       *(
}

void triangulo      :: print () const {
                    cout                <<                          endl                    <<
                    <<                  "Lado 1= "                  <<                      lado1
```

```
                    <<              "Lado 2= "      <<          lado2
                    <<              "Lado 3= "      <<          lado3
}

// clase cuadrado
class               cuadrado        : public        TwoDimensionalShape {
                    double lado     ;
    public:
                    cuadrado        (               double      =
                    void fijar_cuadrado (double);
                    virtual double area () const;
                    virtual void    print() const;
};

cuadrado            ::              cuadrado        (           double
                    fijar_cuadrado  (               l           );
}

void cuadrado       ::              fijar_cuadrado  (           double
                    lado            =               l           >
}

double cuadrado     ::              area            () const {
    return          lado            *               lado        ;
}

void cuadrado       :: print() const {
                    cout            <<              endl        <<
                    <<              "Lado= "        <<          lado
}

////////////////////////////////
// clase ThreeDimensionalShape /
////////////////////////////////

class               ThreeDimensionalShape public                Shape       {
    public:
                    virtual void    print () const=             0           ;
};

// clase cubo
class               cubo            : public        ThreeDimensionalShape {
                    double lado     ;
    public:
                    cubo            (               double      =
                    void fijar_cubo (double);
```

```
                    virtual double area     () const;
                    virtual double volume   () const;
                    virtual void            print() const;
};

cubo                ::                      cubo                (                   double
                    fijar_cubo              (                   l                   );
}

void cubo           ::                      fijar_cubo          (                   double
                    lado                    =                   l                   >
}

double cubo         ::                      area                () const {
    return          6                       *                   lado                *
}

double cubo         ::                      volume              () const {
    return          lado                    *                   lado                *
}

void cubo           :: print() const{
                    cout                    <<                  endl                <<
                    <<                      "Lado= "            <<                  lado
};

// clase paralelepipedo
class               paralelepipedo          : public            ThreeDimensionalShape{
                    double largo            ,                   ancho               ,
    public:
                    paralelepipedo          (                   double              =
                    void fijar_paralelepipedo(                  double              ,
                    virtual double area     () const;
                    virtual double volume   () const;
                    virtual void            print() const;
};

paralelepipedo      ::                      paralelepipedo      (                   double
                    fijar_paralelepipedo    (                   l                   ,
}

void paralelepipedo ::                      fijar_paralelepipedo(                   double
                    largo                   =                   l                   >
                    ancho                   =                   a                   >
                    altura                  =                   h                   >
}
```

```
double paralelepipedo :: area () const {
    return 2 * largo *
}

double paralelepipedo :: volume () const {
    return largo * ancho *
}

void paralelepipedo :: print() const{
                cout << endl <<
                << "Largo= " << largo
                << "Ancho= " << ancho
                << "Altura= " << altura
}

// llama a funcion virtual a partir del apuntador de clase base
// utilizando enlace dinamico
void virtualViaPointer (const Shape * baseCla
                baseClassPtr -> print ();
                cout << endl <<
                << "Volumen= " << baseCla
}

// llama a funcion virtual a partir de referencia a clase base
// utilizando enlace dinamico
void virtualViaReference(const Shape & baseCla
                baseClassRef .print();
                cout << endl <<
                << "Volumen= " << baseCla
}

// funcion principal
void main (){
                clrscr ();
                triangulo t ( 5.2 ,
                virtualViaPointer (& t );
                virtualViaReference ( t );

                cuadrado c ( 8.7 );
                virtualViaPointer (& c );
                virtualViaReference ( c );

                getch ();
                clrscr ();

                cubo cub ( 8.3 );
```

```
                virtualViaPointer   (&   cub   );
                virtualViaReference (    cub   );

                paralelepipedo p    (    4.5   ,
                virtualViaPointer   (&   p     );
                virtualViaReference (    p     );

                getch               ();
}
```