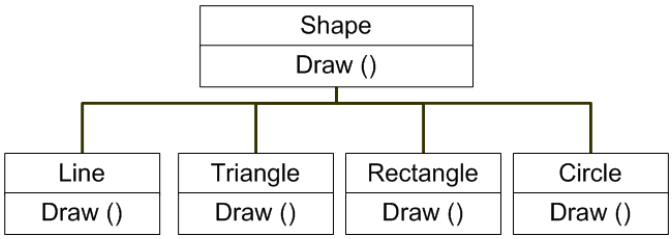
Virtual Functions

Virtual



Virtual functions are the key to **polymorphism**. This allows one function to have different meanings in the parent class as well as the child classes. When you label a function as **virtual**, it is only at run-time that the program knows which function to use. This is known as **late binding**. Here is an example of a parent class and three child classes followed by an explanation:

```
class Parent
{ //Virtual keyword required
    public:
        virtual void display() const
        { cout << “Parent Class Display”; }
};

class Child1 : public Parent
{ //Virtual and override keywords not required but good practice
    public:
        virtual void display() const override
        { cout << “Child1 Class Display”; }
};

class Child2 : public Parent
{ //Functions defined here for brevity (see next page)
    public:
        virtual void display() const override
        { cout << “Child2 Class Display”; }
};

class Child3 : public Parent
{
    //Empty to show that the Parent class' function is invoked
};

int main()
{
    Child1 Obj1;
    Obj1.display(); //Child1 Class Display
    Child2 Obj2;
    Obj2.display(); //Child2 Class Display
    Child3 Obj3;
    Obj3.display(); //Parent Class Display
}
```

The **virtual** keyword is required in the parent class. This sets up the ability to **override** the function that is in the parent class in the child classes. However, the keywords **virtual** and **override** are not required in any of the child classes, but it is good practice to do so. With a virtual function, it is not necessary to override the function as denoted in the **Child3** class.

Please note that the keyword **override** acts as a safety check. It shows the person reading the code that it is indeed a virtual function from the parent class. It also tells the compiler that it is an override and check to make sure the function declaration hasn't been altered.

For brevity, the functions were declared and defined in the class interface. However, they should be defined in the class implementation. When you do this, the keywords **virtual** and **override** are **not** included. For example:

```
void Child1::display() const
{
    cout << "Child1 Class Display";
}
```

The difference between **redefining** a function from the parent class and **overriding** a function from the parent class is that overriding allows for **polymorphism** to take place. Suppose you have this in main:

```
Parent *ParentObject;
Child1 Child1Object;
ParentObject = &Child1Object;
ParentObject->display();
```

This will invoke the display function of the **Child1** class. If this were not a virtual function, then the display function of the parent class would be invoked.

Pure Virtual

Pure virtual functions have the same properties as virtual functions with one exception. They are only implemented in **every** child class. However, the parent class does have to have a declaration of the pure virtual function. This function will have the keyword **virtual** and **= 0** right before the semicolon. That is the distinguishing feature to tell pure virtual functions apart from virtual functions.

A very important note to make is that if you do **not** define a pure virtual function in a child class and create **no** object of said child class, the program will compile. However, it is only when you make an object of said child class that the program errors and will not compile. **Be aware of this!**

Here is an example program implementing a pure virtual function. The keyword **virtual** is not needed in the child classes, but is there for clarity. The keyword **override** is allowed in the child classes, but is not put for brevity.

An **abstract class** is a class in which there is at least **one** pure virtual function. Please keep note that since the parent class is an **abstract class**, you **cannot** create an object of said class and you **cannot** dynamically allocate a child class object into the parent class object. This: **Parent PObject = new Child1;** is illegal.

```
class Parent
{ //This is an abstract class and NO object can be created
  //of this class. Pure virtual functions in the parent class are
  //NOT defined except in ALL child classes.
    public:
        virtual void display() const = 0;
};

class Child1 : public Parent
{
    Public: //(Pure) Virtual function defined here for brevity
        virtual void display() const { cout << “Child1's Function”; }
};

class Child2 : public Parent
{
    Public: //(Pure) Virtual function defined here for brevity
        virtual void display() const { cout << “Child2's Function”; }
};

int main()
{
    Child1 Obj1;
    Obj1.display(); //Displays Child1's display function
    Child2 Obj2;
    Obj2.display(); //Displays Child2's display function
}
```

Just like a virtual function, a pure virtual function should be defined in the class implementation. When you do this, the keywords **virtual** and **override** are **not** included. For example:

```
void Child2::display() const
{
    cout << “Child2's Function”;
}
```