De ce este output-ul gresit si nu cel la care ne asteptam? Functia area() este asociata pointerului shape in momentul compilarii (spre deosebire de momentul rularii). Acest lucru poarta numele de **static linkage** sau **early binding.** Pentru ca pointerul este de tip Shape, va primi functia din clasa de baza.

#include <iostream>

using namespace std**;**

class Shape

**{**

protected**:**

int width**,** height**;**

public**:**

Shape**(** int a **=** 0**,** int b **=** 0**)**

**{**

width **=** a**;**

height **=** b**;**

**}**

int area**()**

**{**

cout **<<** "Parent class area :" **<<**endl**;**

**return** 0**;**

**}**

**};**

class Rectangle**:** public Shape

**{**

public**:**

Rectangle**(** int a **=** 0**,** int b **=** 0**) :** Shape**(**a**,** b**)** **{** **}**

int area **()**

**{**

cout **<<** "Rectangle class area :" **<<**endl**;**

**return** **(**width **\*** height**);**

**}**

**};**

class Triangle**:** public Shape

**{**

public**:**

Triangle**(** int a **=** 0**,** int b **=** 0**) :** Shape**(**a**,** b**)** **{** **}**

int area **()**

**{**

cout **<<** "Triangle class area :" **<<**endl**;**

**return** **(**width **\*** height **/** 2**);**

**}**

**};**

// Main function for the program

int main**()**

**{**

Shape **\***shape**;**

Rectangle rec**(**10**,**7**);**

Triangle tri**(**10**,**5**);**

// store the address of Rectangle

shape **=** **&**rec**;**

// call rectangle area.

shape**->**area**();**

// store the address of Triangle

shape **=** **&**tri**;**

// call triangle area.

shape**->**area**();**

**return** 0**;**

**}**

Cu un singur cuvant cheie ii putem specifica compilatorului sa se uite la tipul variabilei la care arata pointerul nostru si sa invoce functia aferenta acestui tip.

**virtual** int area**()**

**{**

cout **<<** "Parent class area :" **<<**endl**;**

**return** 0**;**

**}**

Practic, cuvantul cheie **virtual** ii spune compilatorului faptul ca nu vrem static linkage pentru aceasta functie ci vrem ca acesta sa se uite de fiecare data la tipul obiectului ce cheama functia, adica un comportament de tipul **dynamic linkage** sau **late binding**.

**Functii pur virtuale**

virtual int area**()** **=** 0**;**

Daca avem un caz similar cu acesta, cand pentru o clasa de baza nu putem sa dam o definitie cu sens unei metode, dar vrem ca aceasta sa existe pentru a o putea mosteni in clase derivate, avem conceptul de functie virtuala. Exista totusi amendamentul ca suntem obligati sa implementam metoda in una din clasele derivate, altfel vom primi o eroare de compilare.

In momentul in care intr-o clasa avem cel putin o functie pur virtuala, clasa devine clasa **abstracta**. O clasa abstracta nu poate fi instantiata, ea este utila doar pentru a servi drept clasa de baza pentru alte clase derivate care se pot instantia dupa ce ii implementeaza functiile pur virtuale.

Totusi, faptul ca nu putem instantia clasa nu inseamna ca nu putem crea pointeri la aceasta pentru a ne bucura de toate facilitatile polimorfismului. Deci exemplul dinainte ramane la fel chiar daca facem area() sa fie pur virtuala.

void printArea**()**

**{**

cout **<<** this**->**area**()** **<<** '\n'**;**

**}**

Adaugati functia aceasta in Shape si schimbati apelurile din main:

shape**->**printArea**();**

Dupa cum vedeti, putem chiar sa folosim pointerul **this** pentru a accesa functia pur virtuala si chiar daca nu avem nici o implementare a acesteia, putem chema metoda area() pentru ca stim ca la momentul apelului, nu vom fi in clasa Shape, ci in una din clasele derivate.

int main**()**

**{**

Shape **\***shape1 **=** new Rectangle**(**10**,** 5**);**

Shape **\***shape2 **=** new Triangle**(**4**,** 6**);**

// call rectangle area.

shape1**->**printArea**();**

// call triangle area.

shape2**->**printArea**();**

**return** 0**;**

**}**

Nu este nevoie sa instantiem obiecte din clasele derivate separat si apoi sa le punem adresa intr-un pointer din clasa de baza.

Am omis in laboratorul trecut sa va spun de suprascrierea functiilor si operatorilor.

#include <iostream>

using namespace std**;**

class printData **{**

public**:**

void print**(**int i**)** **{**

cout **<<** "Printing int: " **<<** i **<<** endl**;**

**}**

void print**(**double f**)** **{**

cout **<<** "Printing float: " **<<** f **<<** endl**;**

**}**

void print**(**const char**\*** c**)** **{**

cout **<<** "Printing character: " **<<** c **<<** endl**;**

**}**

**};**

int main**(**void**)**

**{**

printData pd**;**

// Call print to print integer

pd**.**print**(**5**);**

// Call print to print float

pd**.**print**(**500.263**);**

// Call print to print character

pd**.**print**(**"Hello C++"**);**

**return** 0**;**

**}**

Putem avea functii cu acelasi nume, dar cu antetul functiei diferit. In momentul apelului, compilatorul se uita dupa o functie din interiorul clasei care sa aiba parametrii corespunzatorului apelului nostru.

#include <iostream>

using namespace std**;**

class Box **{**

public**:**

double getVolume**(**void**)** **{**

**return** length **\*** breadth **\*** height**;**

**}**

// Overload + operator to add two Box objects.

Box operator**+(**const Box**&** b**)** **{**

Box box**;**

box**.**length **=** this**->**length **+** b**.**length**;**

box**.**breadth **=** this**->**breadth **+** b**.**breadth**;**

box**.**height **=** this**->**height **+** b**.**height**;**

**return** box**;**

**}**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

**};**

int main**()** **{**

Box Box1**;** // Declare Box1 of type Box

Box Box2**;** // Declare Box2 of type Box

Box Box3**;** // Declare Box3 of type Box

double volume **=** 0.0**;** // Store the volume of a box here

Box1**.**length **=** 6.0**;**

Box1**.**breadth **=** 7.0**;**

Box1**.**height **=** 5.0**;**

Box2**.**length **=** 12.0**;**

Box2**.**breadth **=** 13.0**;**

Box2**.**height **=** 10.0**;**

volume **=** Box1**.**getVolume**();**

cout **<<** "Volume of Box1 : " **<<** volume **<<** endl**;**

volume **=** Box2**.**getVolume**();**

cout **<<** "Volume of Box2 : " **<<** volume **<<** endl**;**

Box3 **=** Box1 **+** Box2**;**

volume **=** Box3**.**getVolume**();**

cout **<<** "Volume of Box3 : " **<<** volume **<<** endl**;**

**return** 0**;**

**}**

Suprascrierea unui operator se precede cu cuvantului cheie operator si simbolul folosit. <https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm> aveti jos operatorii ce pot si nu pot fi suprascrisi.