Polymorphism (1A)

Young Won Lim
8/22/13

Copyright (c) 2011-2012 Young W. Lim.

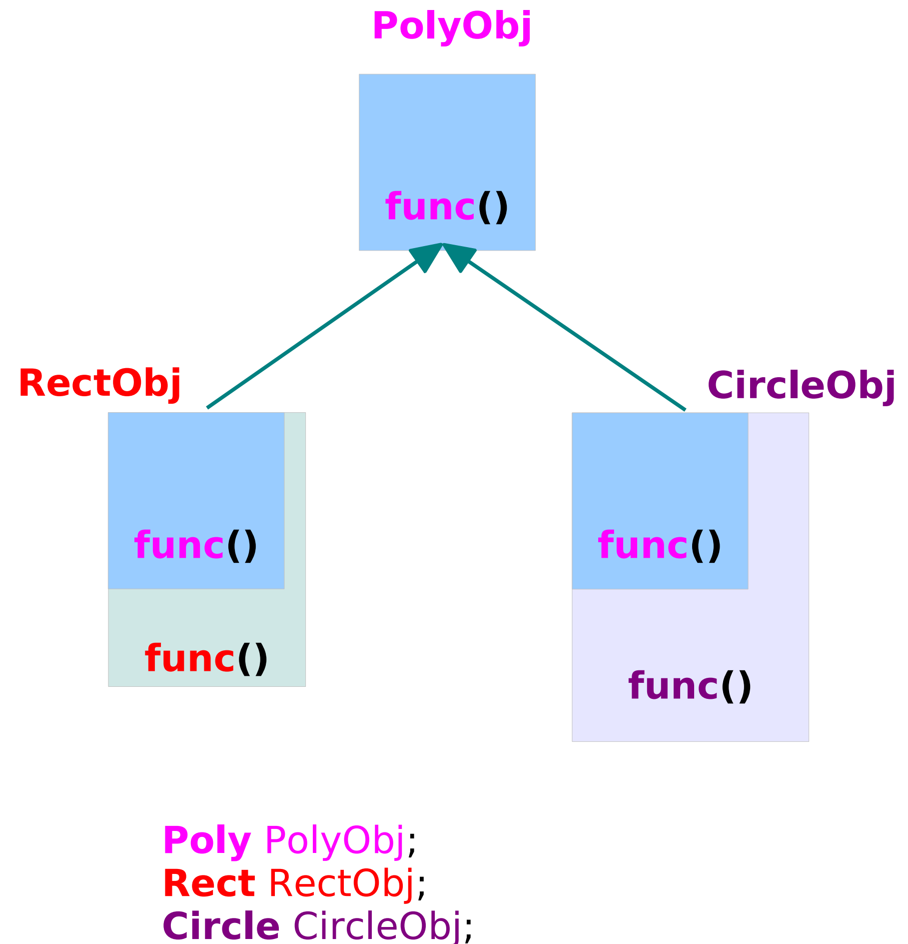
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

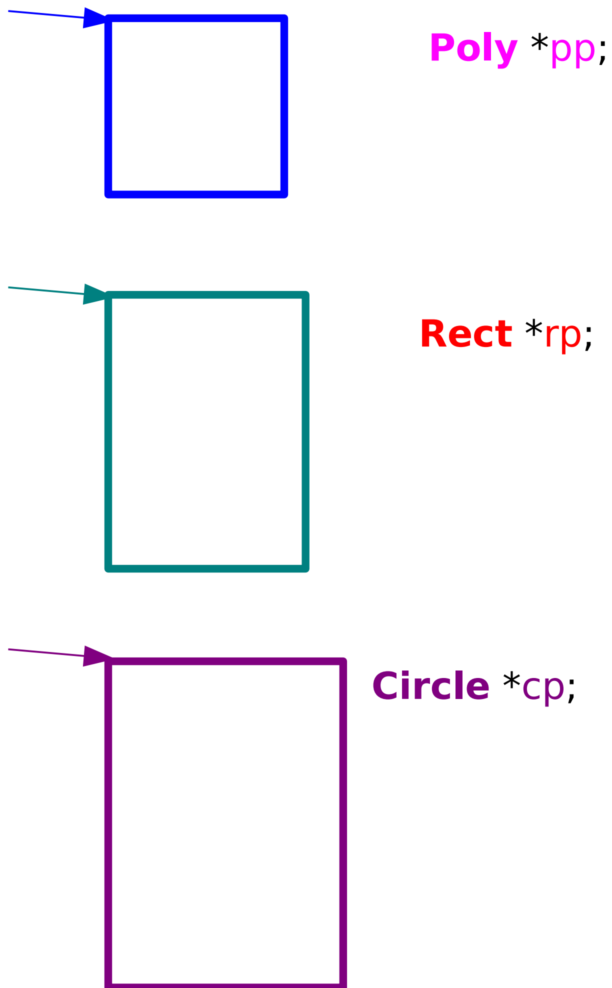
Base & Derived Class Objects

```
#include <stdio.h>

class Poly {
public:
   virtual void func()
  { printf("Poly::func() is called... \n"); }
};

class Rect : public Poly {
public:
   (virtual) void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   (virtual) void func()
  { printf("Circle::func() is called... \n"); }
};
```




```
Poly PolyObj;
Rect RectObj;
Circle CircleObj;
```

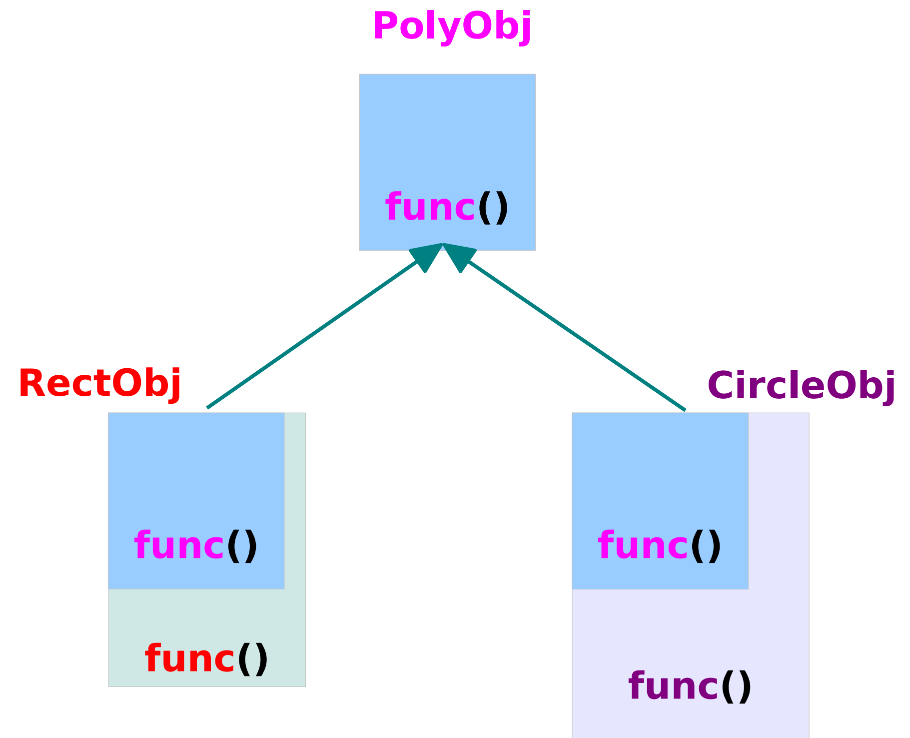
Base & Derived Class Pointers (1)



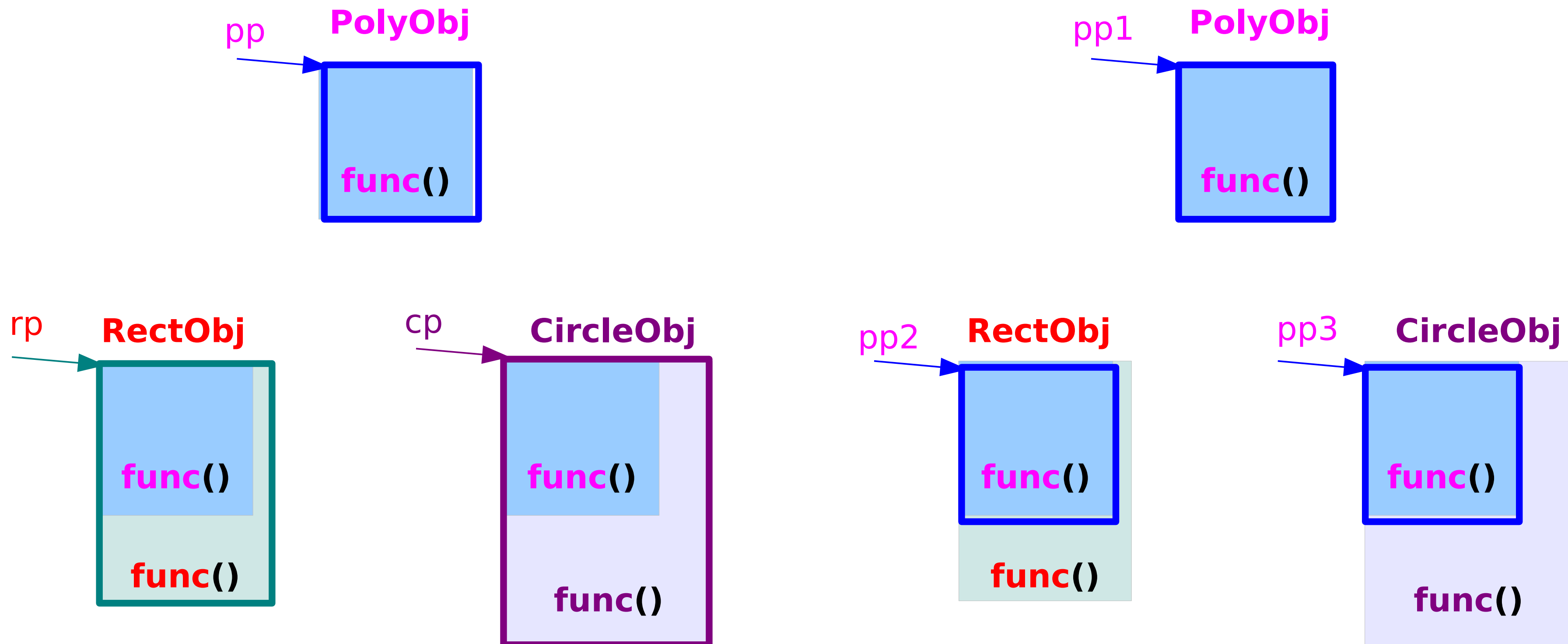
Poly *pp;

Rect *rp;

Circle *cp;



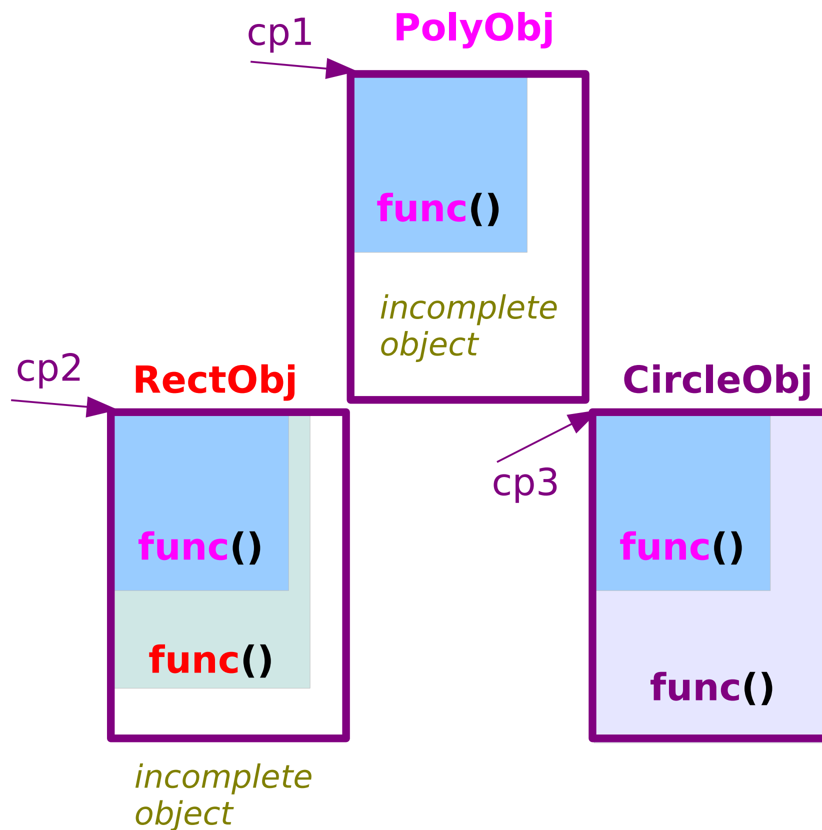
Base & Derived Class Pointers (2)

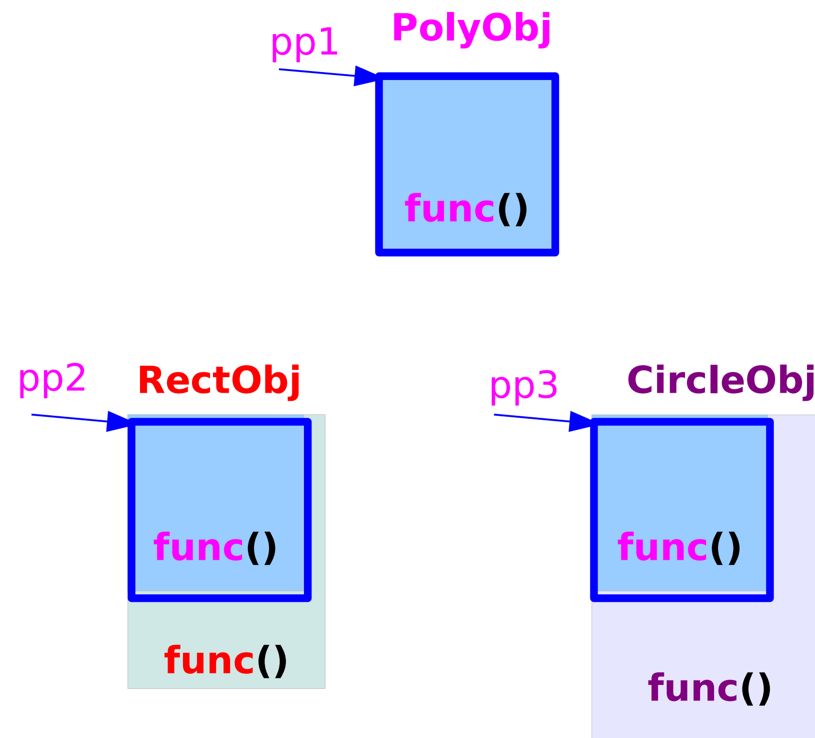



```
pp = &PolyObj;
rp  = &RectObj;
cp  = &CircleObj;
```

```
pp1 = &PolyObj;
pp2 = &RectObj;
pp3 = &CircleObj;
```

Base & Derived Class Pointers (3)






```
cp1 = (Circle *) &PolyObj;
cp2 = (Circle *) &RectObj;
cp3 = (Circle *) &CircleObj;
```

Danger of Run Time Error !!

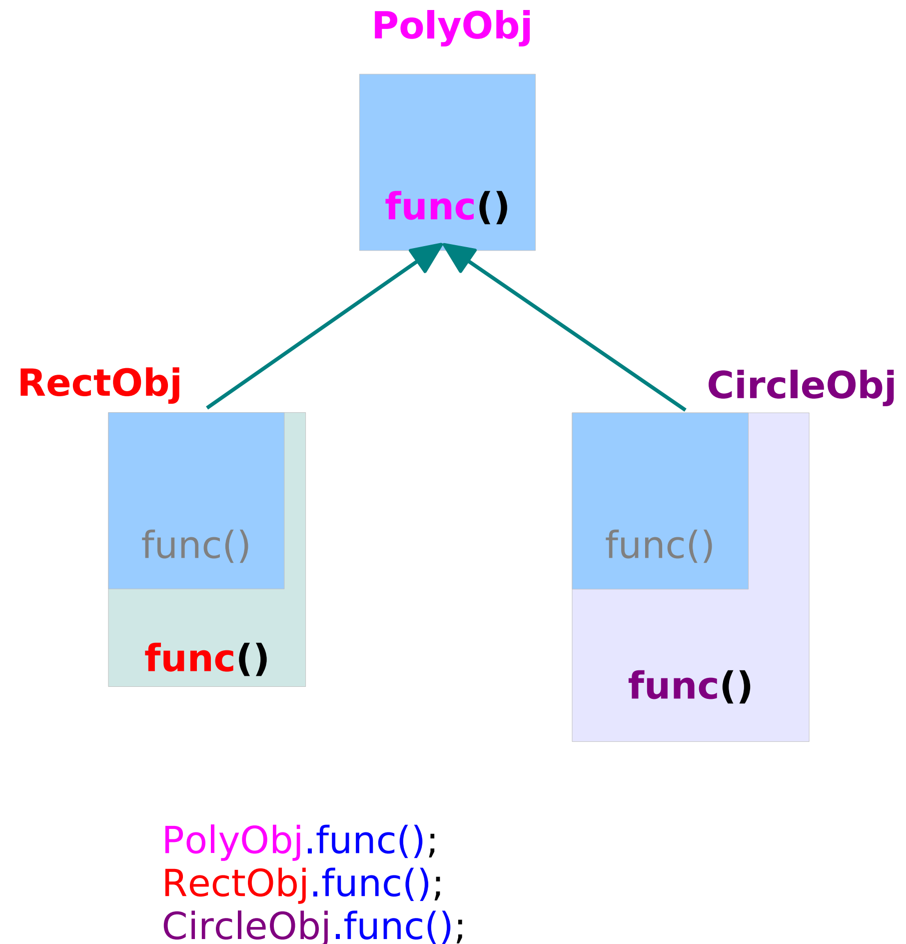
```
pp1 = &PolyObj;
pp2 = &RectObj;
pp3 = &CircleObj;
```

No problem !!

Non Virtual Member Functions (1)

```
#include <stdio.h>

class Poly {
public:
   void func()
  { printf("Poly::func() is called... \n"); }
};

class Rect : public Poly {
public:
   void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   void func()
  { printf("Circle::func() is called... \n"); }
};
```

Hiding base class member functions




```
PolyObj.func();
RectObj.func();
CircleObj.func();
```

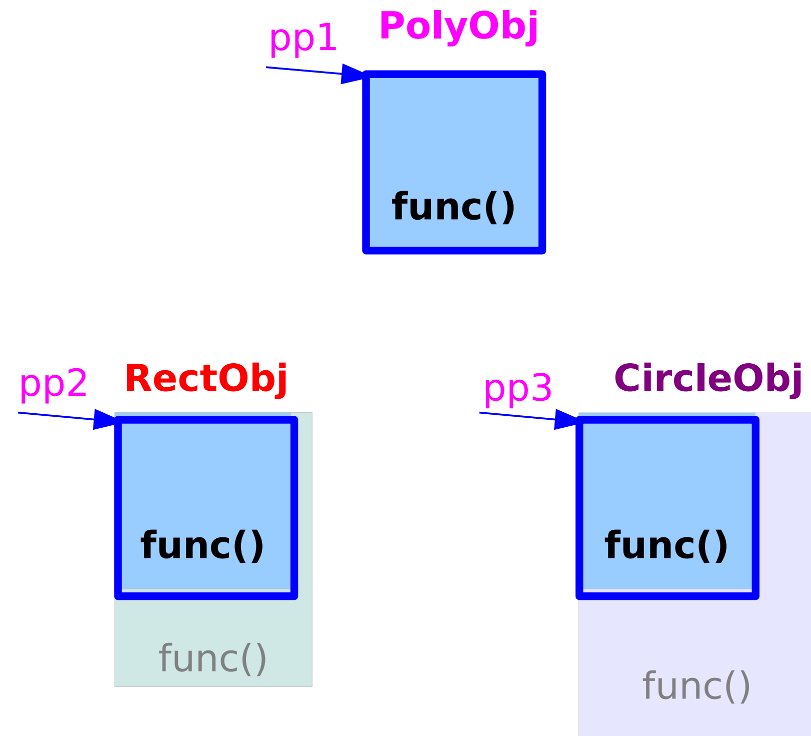
Non Virtual Member Functions (2)

```
#include <stdio.h>
```

```
class Poly {
public:
   void func()
  { printf("Poly::func() is called... \n"); }
};
```

```
class Rect : public Poly {
public:
   void func()
   { printf("Rect::func() is called... \n"); }
};
```

```
class Circle : public Poly {
public:
   void func()
  { printf("Circle::func() is called... \n"); }
};
```




```
pp1 = &PolyObj;
pp2 = &RectObj;
pp3 = &CircleObj;
```

```
pp1->func();
pp2->func();
pp3->func();
```

Poly::func() is called three times

Virtual Member Functions (1)

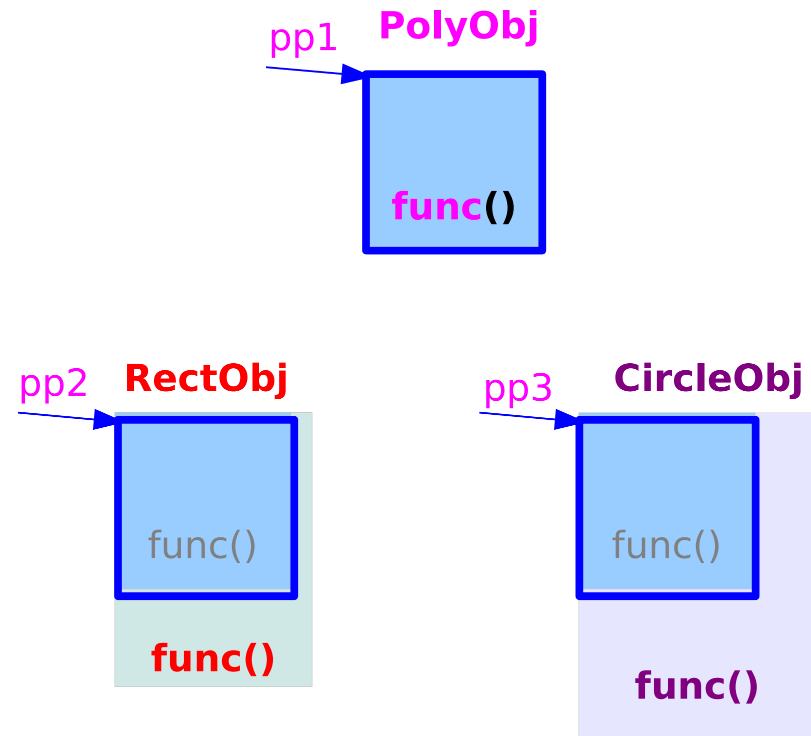
```
#include <stdio.h>
```

```
class Poly {
public:
   virtual void func()
  { printf("Poly::func() is called... \n"); }
};
```

```
class Rect : public Poly {
public:
   (virtual) void func()
   { printf("Rect::func() is called... \n"); }
};
```

```
class Circle : public Poly {
public:
   (virtual) void func()
  { printf("Circle::func() is called... \n"); }
};
```

Virtual functions mechanism:
type dependent function calls at run time
(special run time handling of invocations)
when invoked through a public base pointer




```
pp1 = &PolyObj;
pp2 = &RectObj;
pp3 = &CircleObj;
```

```
pp1->func();
pp2->func();
pp3->func();
```

Virtual Member Functions (2)

```
#include <stdio.h>

class Poly {
public:
   virtual void func()
  { printf("Poly::func() is called... \n"); }
};

class Rect : public Poly {
public:
   (virtual) void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   (virtual) void func()
  { printf("Circle::func() is called... \n"); }
};
```

```
int main(void) {

  Poly PolyObj, *pp;
  Rect RectObj, *rp;
  Circle CircleObj, *cp;

  pp = &PolyObj;
  pp->func();

  pp = &RectObj;
  pp->func();

  pp = &CircleObj;
  pp->func();

}
```

Poly::func()

Rect::func()

Circle::func()

*Without the **virtual** keyword, Poly::func is called 3 times.*

Pure Virtual Member Functions

```
#include <stdio.h>

class Poly {
public:
   virtual void func() = 0;

};

class Rect : public Poly {
public:
   (virtual) void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   (virtual) void func()
  { printf("Circle::func() is called... \n"); }
};
```

```
int main(void) {

  Poly PolyObj, *pp;
  Rect RectObj, *rp;
  Circle CircleObj, *cp;

  pp = &PolyObj;
  pp->func();

  pp = &RectObj;
  pp->func();

  pp = &CircleObj;
  pp->func();

}
```

Rect::func()

Circle::func()

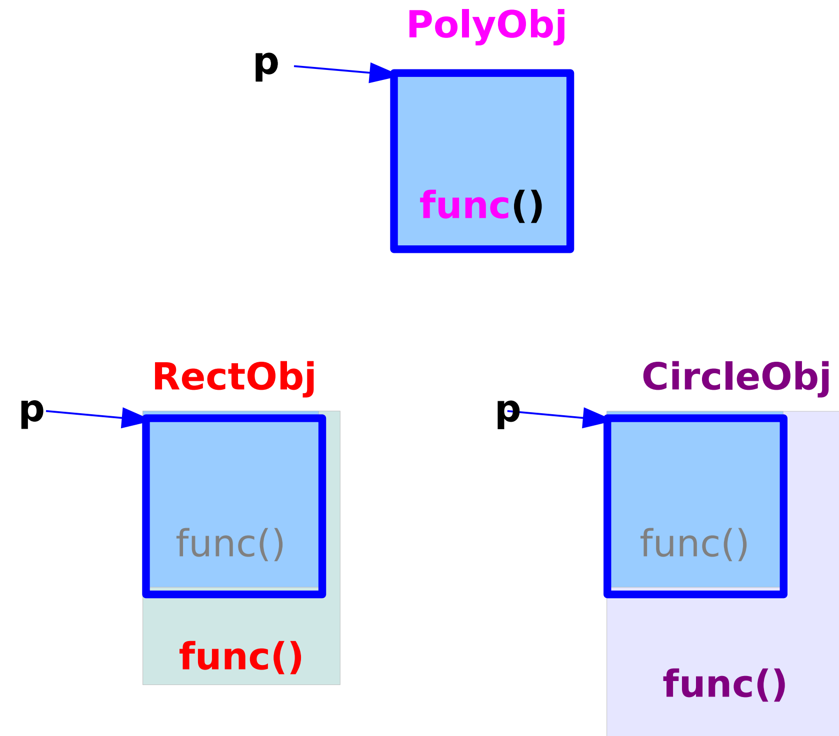
Classes containing pure virtual functions are termed "**abstract**"; they cannot be instantiated directly.

Virtual Function Mechanism

If a virtual function is called
through a public base pointer
then actual type of the object is used
to choose the proper one among a group
of compatible virtual functions

A call to a virtual function is done
through a base reference or pointer
treated specially at run time

Unlike regular functions
a virtual function call involves
choosing one in a group of
compatible virtual functions
according to the actual object type
at run time



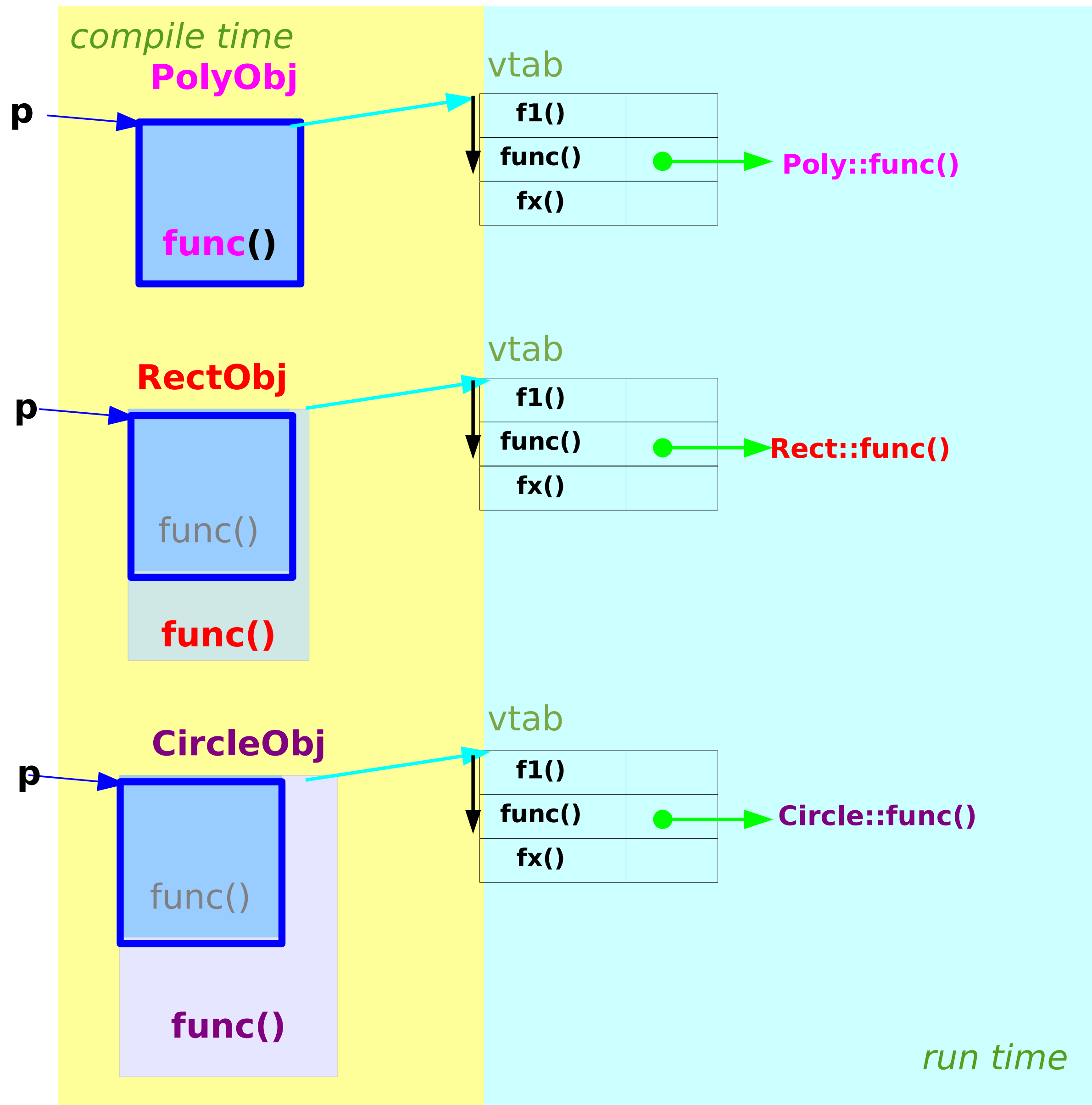

```
void foo(Poly * p) {

    p can be determined at run time

}
```

public base pointer

a group of compatible virtual functions

p->func();	Poly::func()
p->func();	Rect::func()
p->func();	Circle::func()

Virtual Function Table

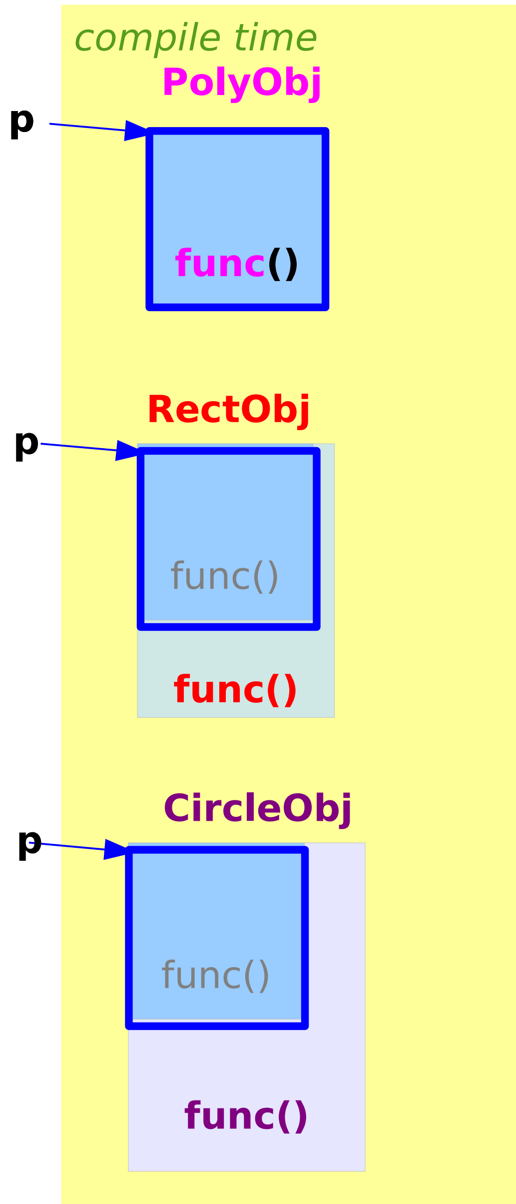



```
void foo(Poly * p) {

    p can be determined at run time

}
```

At run time, the stored address of the vtab pointed by the object to which the base pointer p points, determines the actual address of the function

The compiler translates the virtual function call into the index to the vtab

Run Time Type Identification



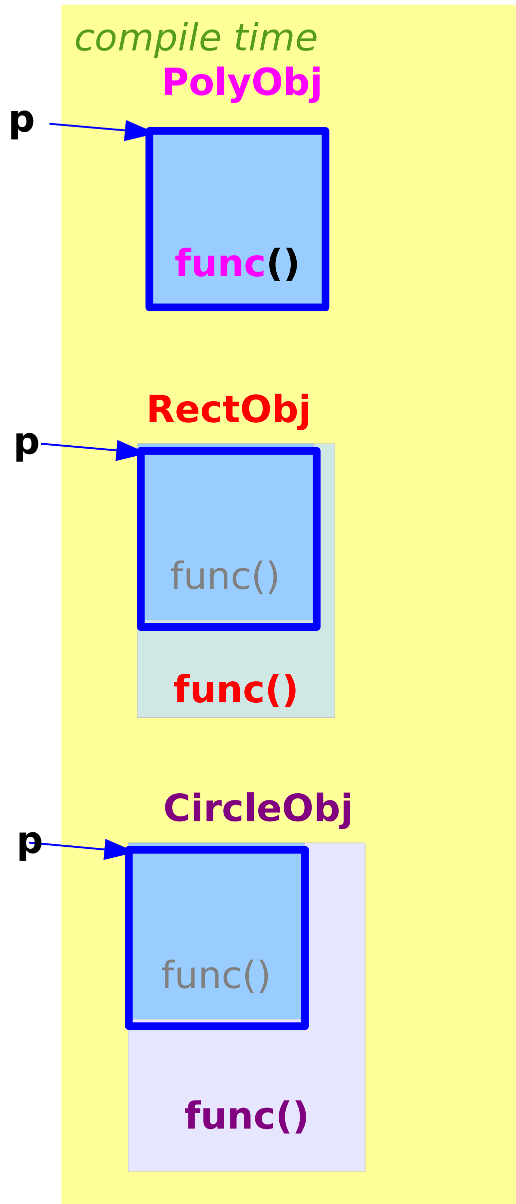
RTTI
Run-Time Type Information
Run-Time Type Identification

a C++ mechanism that exposes information about
an object's data type at runtime.

dynamic_cast

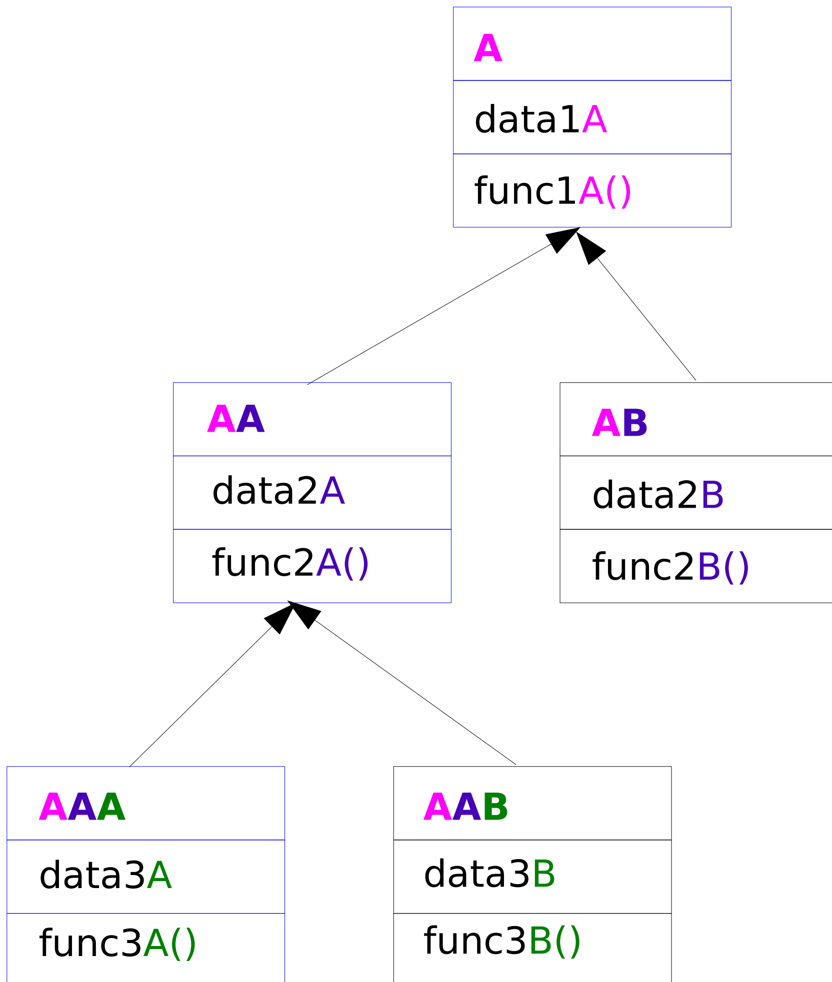
typeid

Dynamic Cast




```
void foo(Poly * p) {

    p can be determined at run time

  Rect   *RectPointer;
  Circle  *CirclePointer;

  RectPointer = dynamic_cast <Rect> (p);

  if (RectPointer != NULL) {

        do specific things pertain to Rect

  }

  CirclePointer = dynamic_cast <Circle> (p);

  if (CirclePointer != NULL) {

        do specific things pertain to Circle

  }

}
```

Class Structure



Class **A**

data1A
func1A()

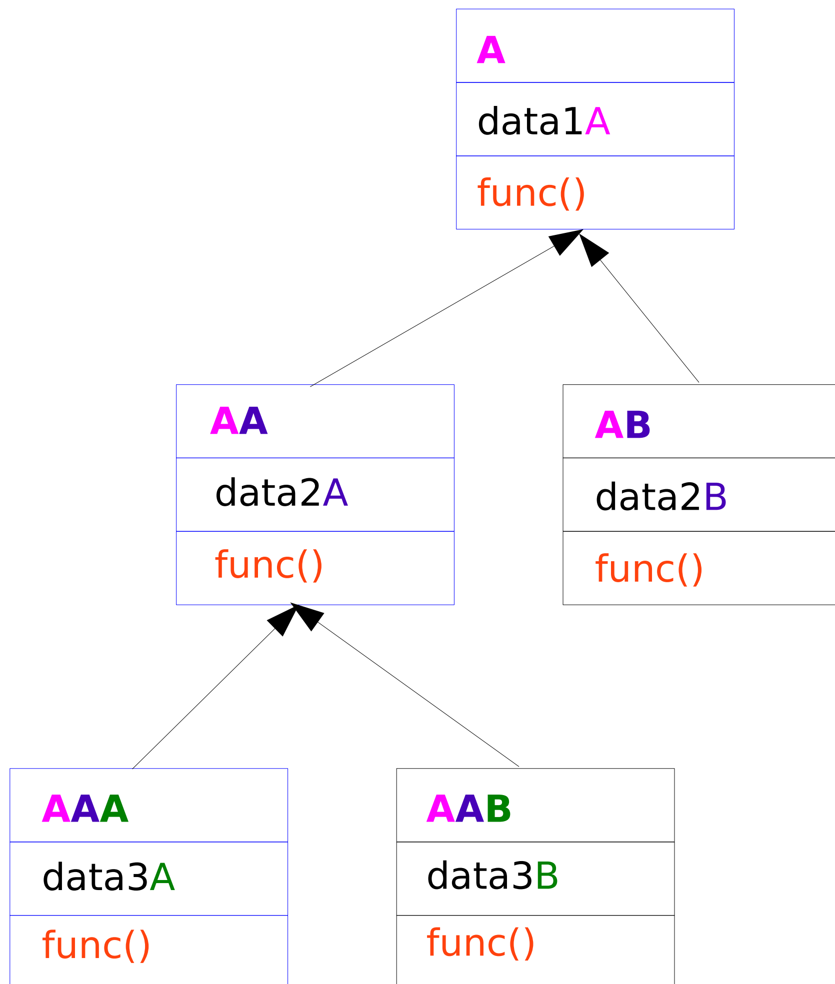
Class **AA**

data1A	data2A
func1A()	func2A()

Class **AA**

data1A	data2A	data3A
func1A()	func2A()	func3A()

Abstract Class (1)



Class A

data1A
func()

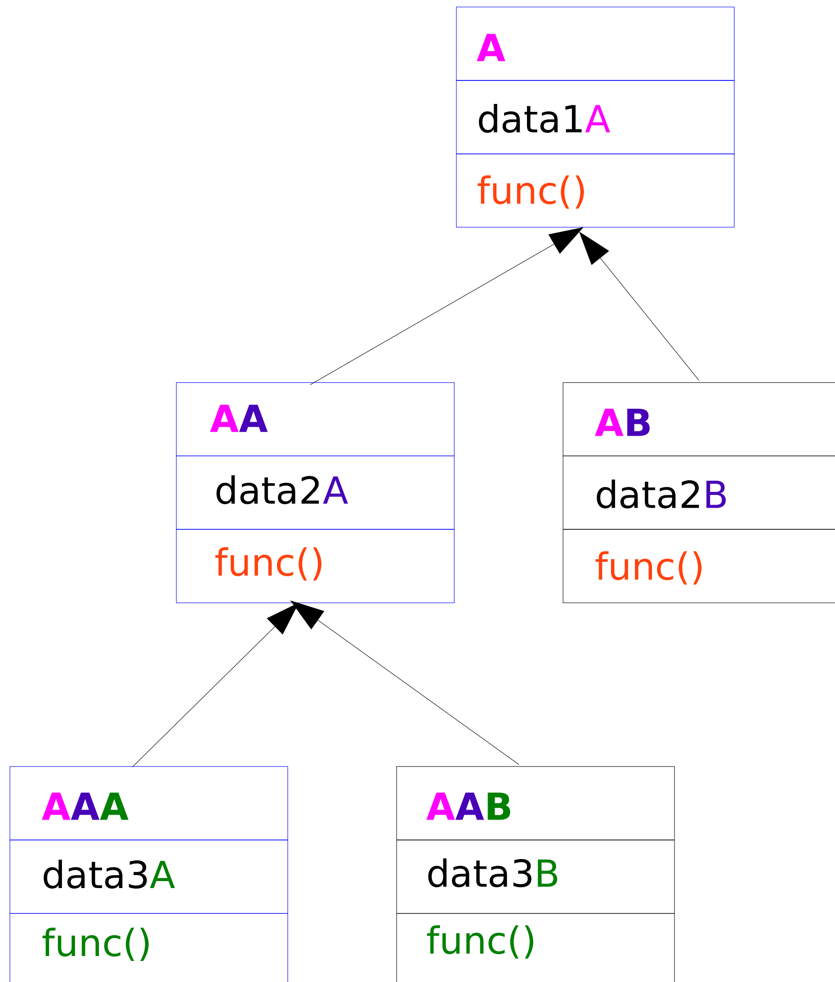
Class AA

data1A	data2A
func()	func()

Class AAA

data1A	data2A	data3A
func()	func()	func()

Abstract Class (2)




```
class A {
public:
        virtual void func() const = 0;
        ...
}

class AA public A {
public:
        virtual void func() const = 0;
        ...
}

class AAA public AA {
public:
        void func() {
                can be defined here.
        }
        ...
}
```

References

[1] Java in a nutshell, 4th ed, David Flanagan
[2] An Introduction to Object-Oriented Programming with Java, C. Thomas, Wu
[3] Power Java, I. K. Chun (in Korean)