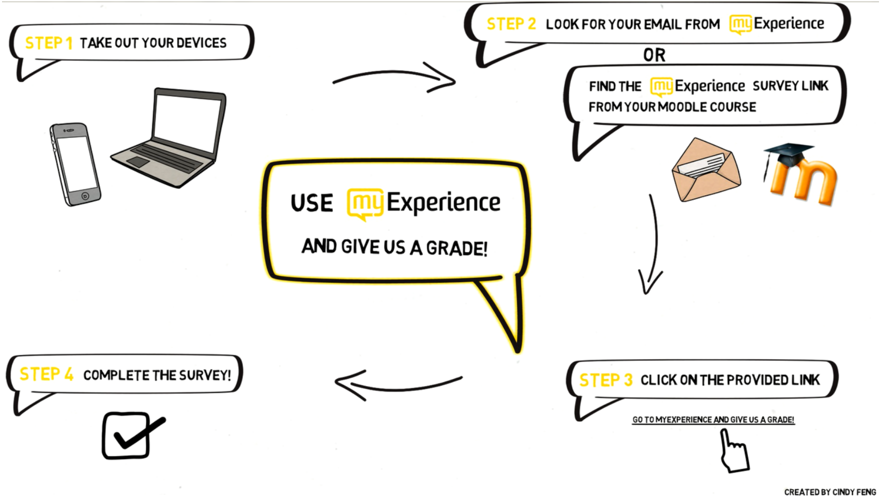
COMP6771
Advanced C++ Programming

Week 11
Object Oriented Programming (Continued)

2016

www.cse.unsw.edu.au/˜cs6771

myExperience Survey



STEP 1 TAKE OUT YOUR DEVICES

STEP 2 LOOK FOR YOUR EMAIL FROM myExperience

OR

FIND THE myExperience SURVEY LINK FROM YOUR MOODLE COURSE

USE myExperience AND GIVE US A GRADE!

STEP 3 CLICK ON THE PROVIDED LINK

GO TO MYEXPERIENCE AND GIVE US A GRADE!

STEP 4 COMPLETE THE SURVEY!

CREATED BY CINDY FENG

Defining a Base Class (for Undiscounted Books)

```
class Book {
public:
  Book (const std::string &book_isbn = "",
              double sales_price = 0.0)
              : isbn_no{book_isbn}, price{sales_price} { }

  std::string isbn() const { return isbn_no; }

  virtual double net_price(std::size_t n) const
             { return n * price; }

  virtual ~Book() { }

private:
   std::string isbn_no;

protected:
   double price;
};
```

Defining a Derived Class (for discounted Books)

```
class Disc_book : public Book {
public:
  virtual double net_price(std::size_t n) const;

private:
   std::size_t min_qty;
   double discount;
};

double Disc_book::net_price(std::size_t n) const {
  if (n >= min_qty)
    return n * (1 - discount) * price;
  else
    return n * price;
}
```

optional, but should use

Some User Code

```
void print_total(std::ostream &os,
                 const Book &b, size_t n) {

  os << "ISBN: " << b.isbn()  // always call Book:isbn()
     << " number sold: " << n << " total price: "
     << b.net_price(n) << std::endl;
};

int main() {
  Book b{"Book 1",9.99};
  Disc_book db; // no inherited constructor
  print_total(std::cout, b, 10);  // call Book::net_price()
  print_total(std::cout, db, 10); // call Disc_book::net_price()
}
```

Static Type and Dynamic Type of Class Objects

- The static or declared type at the declaration
- The dynamic type of the object pointed or referred to

```
--------------------------------------------

                object pointed/referenced by p
declaration              static   dynamic
--------------------------------------------

Disc_book db;
Book *p = &db;            Book   Disc_book
--------------------------------------------

Disc_book db;
Book &p = db;             Book   Disc_book
--------------------------------------------
```

Static and Dynamic Typing/Binding

- C++:
  - Statically typed
  - Static binding for nonvirtuals (based on static type of receiver)
  - Dynamic binding for virtuals (based on dynamic type of receiver)
- Java:
  - Statically typed
  - Dynamic binding only (all functions are virtual)
- Dynamically typed languages: Smalltalk and APL
- For the pros and cons of static and dynamic typing, see:

  `Robert W. Sebesta, Concepts of Programming Languages, 10th Ed, Addison-Wesley.`

Static: compile-time
Dynamic: run-time

Static Binding for Nonvirtuals

```
Book b;
Disc_book db;
Book *p = &b, *q = &db;
```

- `p->isbn();`
  - Static typing: static type of p is Book
  - Static binding: call Book::isbn()
- `q->isbn();`
  - Static typing: static type of q is Book
  - Static binding: call Book::isbn()
- Similarly if p and q are references

  ```
  Book &p = b, &q = db;
  ```

Dynamic Binding for Virtuals

- Can be achieved with pointers to base classes:

```
Disc_book db;
Book *pb = &db;
Disc_book *pd = &db;
pb->net_price(); // call Disc_book::net_price()
pd->net_price(); // call Disc_book::net_price()
```

- Static typing:

```
   pb->net_price(); // Book::net_price() exists
   pd->net_price(); // Disc_book::net_price() exists
```

- Dynamic binding: runtime resolution of the function called based on the dynamic type of the object
- net_price is said to be polymorphic since we can also have:

```
Book b;
Book *p = &b;
p->net_price() // call Book::net_price()
```

- Can also be achieved via references to base classes

Data Types Revisited

- A type: a set of values and a set of operations on the values
- Book:
  - set of values: Book, Disc_book objects, ...
  - set of operations, e.g., Book::isbn()
    - The interface of Book is still the same as before
    - But its implementation can be changed in Disc_book
- Polymorphic type: a type with virtual functions
- Note: static typing works as before

Pure Virtual Functions

- A class is an abstract base class (ABC) if it contains some pure virtual functions (declared or inherited)

```
class shape {
public:
   virtual void draw() = 0;    // pure virtual
};

class circle : public shape {
   void draw() { /* draw a circle */ }
};

class square: public shape {
   void draw() { /* draw a square */ }
};
```

- ABC's are similar to Java's interfaces
- No objects can be constructed from ABC's

C++ Member Functions

Syntax	Name	Conceptual Meaning
virtual void draw()=0	pure virtual	inherit interface only
virtual void net_price()	virtual	inherit interface & an optional impl.
std::string isbn()	nonvirtual	inherit interface & a mandatory impl. invariance over specialisation — can be broken immorally with name hiding

(Representative) C++ Object Model

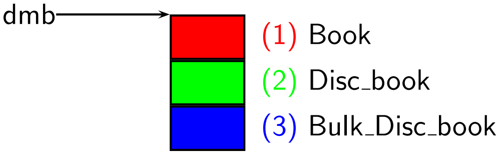
- What are not represented in an object:
  - Static data members
  - Static member functions (static binding)
  - Nonvirtual functions (static binding)
- What are represented in an object:
  - Nonstatic data members
  - Virtual functions

C++ Object Model

- Each polymorphic class has a virtual table, called vtable, containing the addresses for all the virtual functions
- Each object of such a class has a hidden virtual pointer, called vptr, pointing to the beginning of its vtable
- The compiler stores the data members of a class in some predefined order, say, from the least to most derived base class, and finally, the data members in the class itself
- The compiler also chooses the order for the data members in each class, typically, in the order of declaration

Memory Layout for Data Members

```
class Book {...};
class Disc_book: public Book {...};
class Bulk_Disc_book : public Disc_book {...};
Bulk_Disc_book dmb;
```



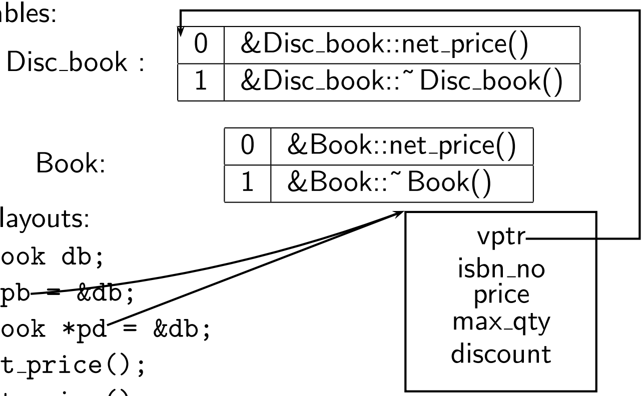
- Subobjects
- Data members in a subobject stored, say, in declaration order

Memory Layout for Virtual Functions (i.e., vtables)

```
class A {                 vtable for A: [0 | &A:f(int) ]
public:                                 [1 | &A:g()    ]
        virtual void f(int);
        virtual int g();
};
class B : public A {      vtable for B: [0 | &B:f(int) ]
public:                                 [1 | &A:g()    ]
        virtual void f(int);            [2 | &B:h()    ]
        virtual void h();
};
class C : public B {      vtable for C: [0 | &B:f(int) ]
public:                                 [1 | &A:g()    ]
        virtual int x();                [2 | &B:h()    ]
};                                      [3 | &C:x()    ]
```

vtables – vptr Initialized in Constructors

- The vtables:



Disc_book :

0	&Disc_book::net_price()
1	&Disc_book::~Disc_book()

Book:

0	&Book::net_price()
1	&Book::~Book()

- Object layouts:

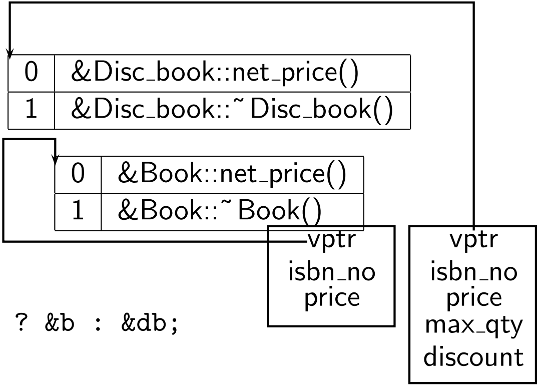
```
Disc_book db;
Book *pb = &db;
Disc_book *pd = &db;
pb->net_price();
pd->net_price();
```

- The compiler generates code to call Disc_book::net_price() in both cases (dynamic binding)

```
1. Load the address (&) of net_price from
   the vtable of the pointed-to object
2. Call the function at the address
```

vtables

- Object layouts:



Disc_book :

0	&Disc_book::net_price()
1	&Disc_book::˜Disc_book()

Book:

0	&Book::net_price()
1	&Book::˜Book()

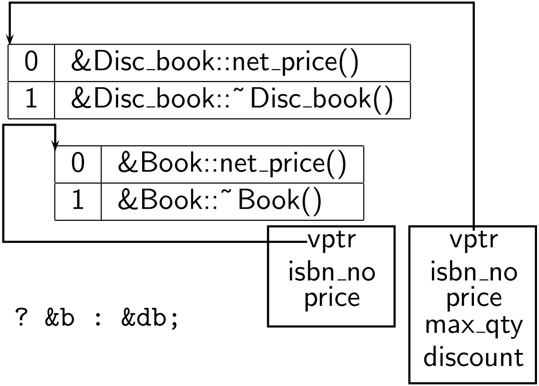
```
Book b;
Disc_book db;
Book *p = (x > y) ? &b : &db;
p->net_price();
```

- The compiler generates the same code, which will call the appropriate net_price() since the same offset for both implementations of net_price is used in both vtables!

```
1. Load the address (&) of net_price from
   the vtable of the pointed-to object
2. Call the function at the address
```

Dynamic Binding Under the hood

- Object layouts:



Disc_book :

0	&Disc_book::net_price()
1	&Disc_book::˜Disc_book()

Book:

0	&Book::net_price()
1	&Book::˜Book()

vptr
isbn_no
price

vptr
isbn_no
price
max_qty
discount

```
Book b;
Disc_book db;
Book *p = (x > y) ? &b : &db;
p->net_price();
```

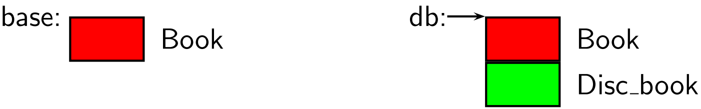
- The code for `p->net_price();`, i.e., `net_price(&p);`

```
push  si        // si contains &(*p)
mov   bx, word ptr [si] // get vtable
call  word ptr [bx+0]   // 4 bytes for address
add   sp, 4   // clean up the stack
```

Object Slicing Revisited

- A nuisance in C++ and should be avoided

```
Disc_book db;
Book b = db;
```



- the vptr in base points now to the vtable for Book
- the vptr for b is initialised in the copy constructor
- Polymorphism achieved only via pointers and references

C++'s Name Hiding Rule

```
class Fruit {
public:
  virtual void eat(float f) { std::cout << "F::e, "; }
};

class Apple : public Fruit {
public:
  virtual void eat(int i) { std::cout << "A::e, "; }
};

int main() {
  Apple *a = new Apple();
  Fruit *f = a;
  f->eat(3.14F);
  a->eat(3.14F);
}
```

What's the Output?:

(A) F::e, F::e, (B) F::e, A::e, (C) A::e, F::e, (D) A::e, A::em,

C++'s Name Hiding Rule

Answer: (B) (B) F::e, A::e,

f is hidden in the base class

- Overloading: same scope, same name but different signatures (return types ignored)
- Overriding: same name, same signature but different scopes (covariant return types)

Advice on Handling C++'s Hiding Rule

- Avoid name hiding if possible
- Can fix the previous example as follows:

```
class Fruit {
public:
  virtual void eat(float f) { std::cout << "F::e, "; }
};

class Apple : public Fruit {
public:
  using Fruit::eat;
  virtual void eat(int i) { std::cout << "A::e, "; }
};

int main() {
  Apple *a = new Apple();
  Fruit *f = a;
  f->eat(3.14F);
  a->eat(3.14F);
}
```

Output is now: F::e, F::e,

Advice on Handling C++'s Hiding Rule

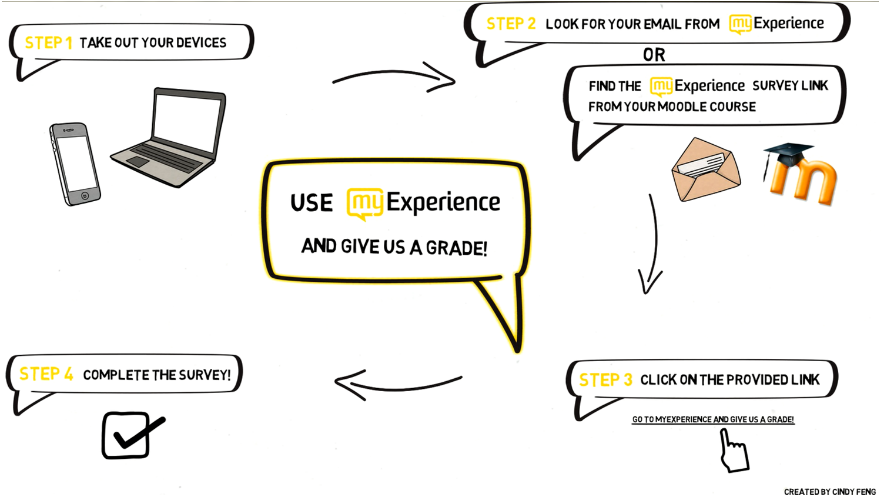
- Avoid name hiding if possible
- Can also fix it as follows in C++11:

```
class Fruit {
public:
  virtual void eat(float f) { std::cout << "F::e, "; }
};

class Apple : public Fruit {
public:
  virtual void eat(int i) override { std::cout << "A::e, "; }
};

int main() {
  Apple *a = new Apple();
  Fruit *f = a;
  f->eat(3.14F);
  a->eat(3.14F);
}
```

- The compiler will complain about override:

```
Fruit.cpp:11:16: error: virtual void Apple::eat(int)
marked override, but does not override
```

myExperience Survey


STEP 1 TAKE OUT YOUR DEVICES
STEP 2 LOOK FOR YOUR EMAIL FROM myExperience
OR
FIND THE myExperience SURVEY LINK FROM YOUR MOODLE COURSE
USE myExperience AND GIVE US A GRADE!
STEP 3 CLICK ON THE PROVIDED LINK
GO TO MYEXPERIENCE AND GIVE US A GRADE!
STEP 4 COMPLETE THE SURVEY!
CREATED BY CINDY FENG