316 11-12-09

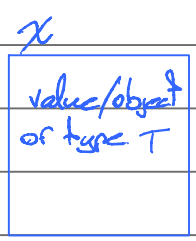
Note Title 11/12/2009

Memory Mapping of Class Objects in Statically Typed OOPLs

C++, C#, Java, Eiffel, etc.

Let x be a variable declared to be of type T. {size(T) = size of the memory cell allocated to one element of T}

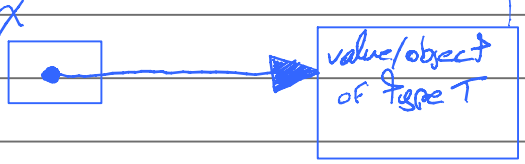
① T is of non-reference type



② T is of reference type

usually inside heap (eg. dynamically created by constructor functions)

size(x) = size(T)



size(T) = the size of the memory cell allocated to a reference itself

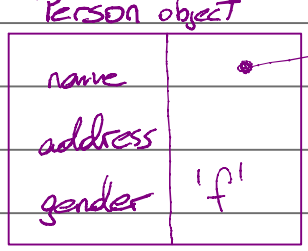
In Java/Eiffel, all class types (including arrays types) belong to category 2

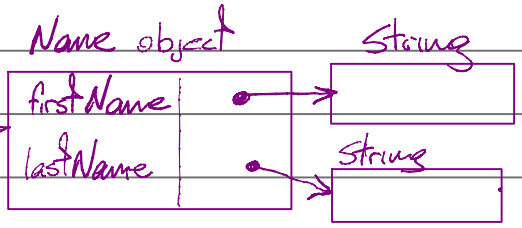
In C++, class types can be either 1 or 2.

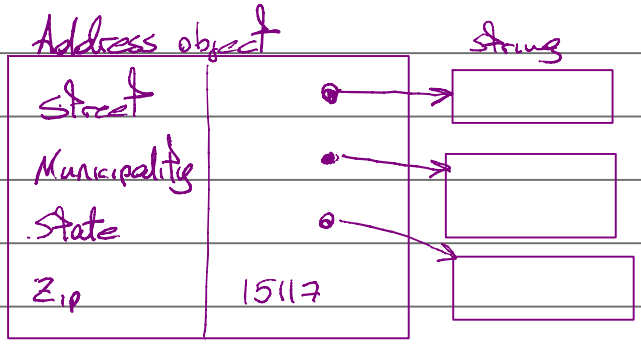
```
C x; //non-reference type
C* x; //reference type
```

Java / C++ with *

```
class Person
{
    Name name;
    Address address;
    char gender;
}
class Name
{
    String firstName;
    String lastName;
}
class Address
{
    String street;
    String municipality;
    String state;
    int zip;
}
```







C++ without ★

Person object

name.firstName
name.lastName
⋮

Memory cell format for class objects without Dynamic Binding of function code

(No inheritance polymorphism)

Base Address → x_1

x_2

⋮

x_m

- record structures in
- ADA, Modula, etc.
- struct type in C

```
class X
{ T1 x1;
  ⋮
  Tm xm;
  f1(...);
  ⋮
  fn(...);
}
```

m instance level fields.

The compiler will call the code for f_j as it is defined in class X

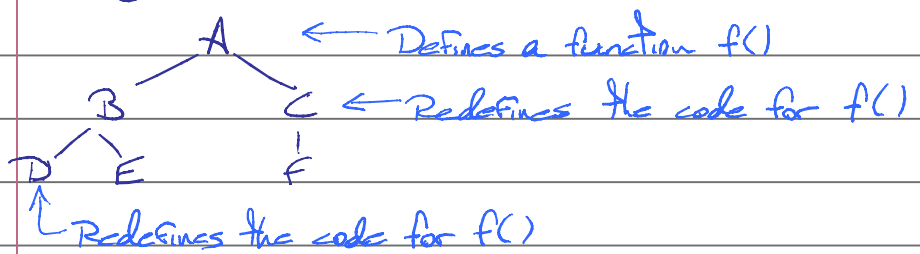
n instance level method functions

The functions f_j are statically bound at compilation time

$$address(x_i) = BaseAddress + \sum_{1 \le k \le i-1} size(x_k), \quad 1 \le i \le m$$

With Dynamic Binding of Function Code

Single-Inheritance Tree



A a; // Java, Eiffel, etc. - dynamic binding is default

A★ a; // C++ - dynamic binding is enabled when reference type is used and f() is declared to be a virtual function

a.f(); // a may refer to A-object or object of any subclass of A

a refers to A-object ⇒ f() defined in A will be called

a refers to B-object ⇒ f() defined in A will be called

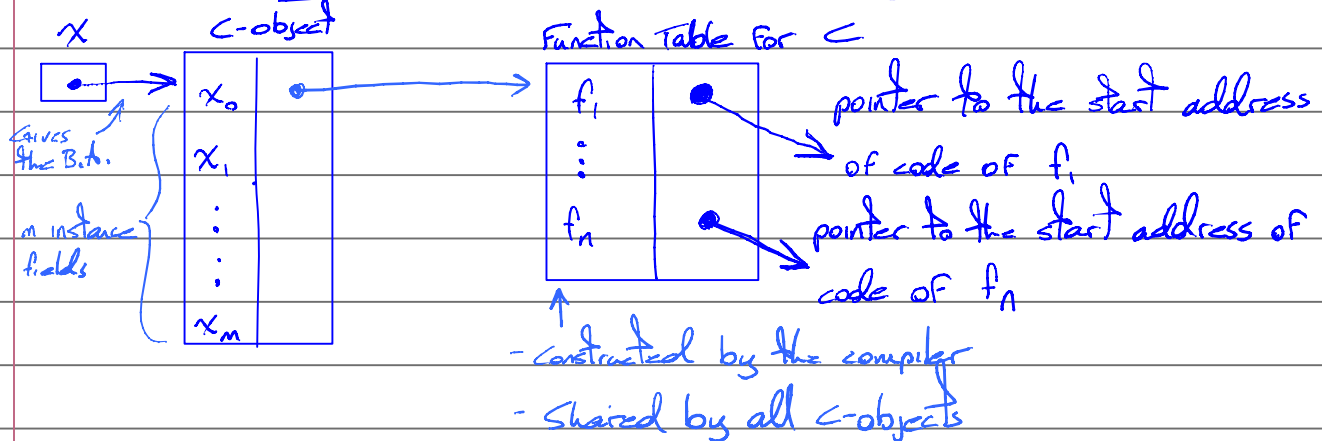
a refers to C-object ⇒ f() defined in C

a refers to D-object ⇒ f() defined in D

a refers to E-object ⇒ f() defined in A

a refers to F-object ⇒ f() defined in C

Use of function tables (method tables) is most common



address(x_i) = BaseAddress(reference value in x) + offset(i)

$= \text{BaseAddress}(\text{reference value in } x) + \sum_{0 \le k \le i-1} \text{size}(x_k), \quad 0 \le i \le m$

address(f_j) = Reference in x_0 field + offset(j)

$= \text{Reference in } x_0 \text{ field} + \sum_{1 \le k \le j-1} \text{size}(f_k), \quad 1 \le j \le n$

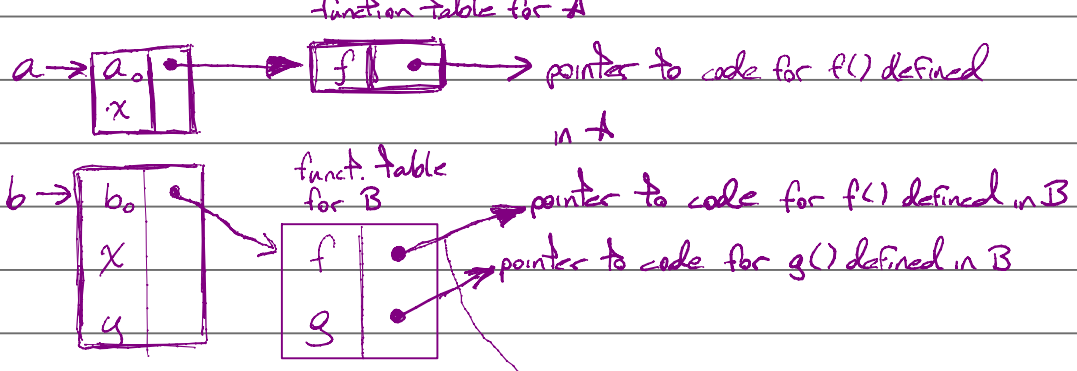
Use Java-like Notation

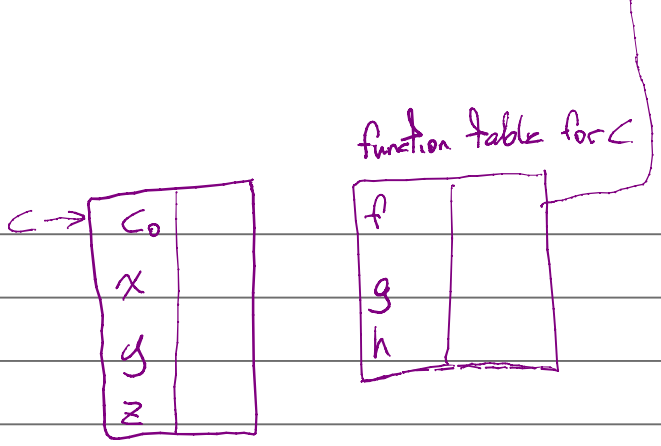
```
class A                          A
{                                |
  int x;                         B
  void f() {...}                 |
}                                C

class B extends A
{
  int y;
  void f() { /* f redefined */ }
  void g() {...}
}

class C extends B
{
  int x;
  void g() { /* g redefined */ }
  void h() {...}
}

void main()
{
  A a = new A();
  B b = new B();
  C c = new C();
}
```




function table for C
C
C_0
x
y
z
f
g
h