CSE 374
Programming Concepts & Tools

Hal Perkins

Fall 2015

Lecture 21 – Function Pointers and Objects in C

Function pointers

- "Pointers to code" are almost as useful as "pointers to data". (But the syntax is painful in C.)
- (Somewhat silly) example:

```
void app_arr(int len, int * arr, int (*f)(int)) {
   for(int k = 0; k < len; k++)
          arr[k] = (*f)(arr[k]);
}
int twox(int i) { return 2*i; }
int sqr(int i)    { return i*i; }
void twoXarr(int len, int* arr) {app_arr(len,arr,&twox);}
void sqr_arr(int len, int* arr)  { app_arr(len,arr,&sqr); }
```

C function-pointer syntax

- C syntax: painful and confusing. Rough idea: The compiler "knows" what is code and what is a pointer to code, so you can write less than we did on the last slide:

```
arr[k] = (*f)(arr[k]);
  ⇒ arr[k] = f(arr[k]);
app_arr(len,arr,&twox);
  ⇒ app_arr(len,arr,twox);
```

- Examples: Compute integral with function (pointer) to integrate and bounds as parameters (int1.c, int2.c)

What is an object?

First Approximation

- An object consists of data and methods
  - Provides the correct (conceptual) model
  - Easy to explain

- But…
  - Doesn't make engineering sense — we don't want to replicate the (same) method bodies (function code) in every object

What is an object?

Second Approximation

- An object consists of data and pointers to methods
- The compiler adds an additional, implicit "this" parameter to every method holding a reference to the receiver object
  - Gives the method a way to refer to the instance variables of the correct receiver object
  - Actual method (function) code has no other connection to any particular object
- Avoids code duplication
- See BAccount1.c (C version of BAccount.cpp)

But. . .

- Still wastes space for pointers to every class function in every object, particularly if there is relatively little instance data, or if the class has a large number of methods

What is an object?

How it's really done (C++, Java, et al):

- There is a single "virtual function" table (vtable) for each class containing pointers to the methods of that class.
  - This is static, constant class data – does not change during execution; initialized at load/startup time
- An object consists of data and a pointer to its class vtable
- Method calls are indirect through the vtable
- Each method still has an implicit *this* parameter that refers to the receiving object
- Avoids code duplication
- Avoids method pointer duplication
- Costs an indirect pointer lookup during each function call
- Example: BAccount2.c

Inheritance and overriding

Basic ideas:

- We have a vtable for every class and subclass
- The vtable for a subclass points to the correct methods — either ones belonging to the base class that are inherited, or ones belonging to the subclass (added or overriding)
- Key idea: The initial part of the vtable for a subclass points to the methods that are inherited or overridden from the base class in exactly the same order they appear in the base class vtable
  - So compiled code can find the correct method at the same offset in the vtable whether it is overridden or not
- Use casts as needed to adjust references up and down the inheritance chain