CSE 333
Systems Programming

Hal Perkins

Winter 2016

Bonus Lecture– Function Pointers and Objects in C

Reminders

- Project due tomorrow night, 11 pm

- Final exam Wed., 2:30-4:20
  - Review & wrapup in section tomorrow
  - Last-minute Q&A Tue. 4:30, GUG 218
  - Topic list + old exams on the web
    - Biased towards stuff since the midterm, but everything is fair game

Agenda

- Function pointers in C/C++ (review/reminder)
- Objects in C – what is “this” anyway?
- Objects in C – virtual functions / dynamic dispatch

Function pointers (reminder)

- “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);
```

- A function pointer in C/C++ is just the address of the first instruction of the function body
- Typedefs make function-pointer declarations less painful
- Examples: Compute integral with (pointer to) function to integrate and bounds as parameters (int1.c, int2.c)

Objects in C++

- What is an object?
  - Simplest answer: a collection of data and functions (methods) to provide behavior
  - Methods can reference instance variables as simple names if unambiguous, or as this->name (always)

  - see thing1.cc, thing2.cc
    - Only non-virtual (static dispatch) for now

So what is “this” anyway?

- In C++ **`this`** is a pointer to the current object when a member function is called
- If the object has type T, “this” has type T*
- But how does it really work? There are no “this” pointers in the x86-64 instruction set…
- Answer: the compiler translates member functions to ordinary x86-64 code, and adds an implicit, hidden “this” parameter to every member function definition and call

Source-level view

What we say to dogs
Okay, Ginger! I've had it! You stay out of the garbage! Understand, Ginger? Stay out of the garbage, or else!
What they hear
blah blah GINGER blah blah blah blah blah blah blah GINGER blah blah blah blah blah blah...

- What you write:

```
int getX() {
  return x_
}
void setX(int x) {
  x_ = x;
}
…
n= t1.getX();
t2->setX(333);
```

- What you really get:

```
getX(Thing *this) {
  return this->x_
}
void setX(Thing *this, int x) {
  this->x_ = x;
}
…
n= t1.getX(&t1);
t2->setX(t2, 333);
```

See thing.c

What is an object, really?

- Methods (behavior, functions) + state (instance vars)
- Actual representation (per object)
  - pointer to class vtable
  - state (instance vars)
- Vtables
  - One per class
  - Pointers to all virtual methods for that class (either inherited or overridden/added by class)
- Virtual function call – indirect through vtable
- Non-virtual function call – resolved using static type of variable that references the object

Compiling obj.m(arguments)

1. Determine (static) type of obj from variable declaration or expression type. Call it T.
2. Verify that type T has a suitable method m with correct number and types of parameters.
   - If more than one such method use overloading rules to pick correct one. Reject as ambiguous if no unique “best” match.
3. Generate function call
   - If method m is not virtual, call T::m
   - If method m is virtual, call m indirectly via vtable pointer in obj (obj->vtbl->m(args))

Examples

- widget.cc – C++ code with class, derived class, and mix of virtual and non-virtual functions
- widget.c – same program in C with explicit vtables (structs with pointers to functiosn) and vtable pointers in objects