CSCI 334:
Principles of Programming Languages

Lecture 18: C/C++

Instructor: Dan Barowy

Williams

Announcements

Homework help session will be tomorrow from 7-9pm in Schow 030A instead of on Thursday.

Announcements

HW6 and HW7 solutions

Announcements

We only have three weeks of class left!

Start thinking about the final exam now.

I am still happy to meet privately with anyone who wants to review their midterm with me.

C Features

- no memory abstraction
  - pointers
    - a pointer is not a data type; it's just an int!
    - operations
      - "address of" operator: &
        - takes any *variable* and returns its *memory address* (i.e., pointer)
      - "dereference" operator: *
        - takes any *pointer* and returns the *value* at that *memory address*
      - "member selection" operator: .
      - "pointer member selection" operator: ->
        - p->foo equivalent to (*p).foo

C Features

- Separate compilation
  - C does not have a "module system"
  - How does C find `printf` below?

```
#include <stdio.h>

int main(int argc, char** argv) {
  printf("Hello world!\n");
}
```

- statements of the form `#<command>` are *preprocessor directives*
  - The C preprocessor is a programmable copy and paste tool

C Features

```
$ clang -E helloworld.c > helloworld_pre.c
```

```
… support code …

#include <stdio.h>

int main(int argc, char** argv) {
  printf("Hello world!\n");
}
```

- (demo)

C Features

Typedef (demo)

Activity

Write a swap() function with the following function header:

```
void swap(int *p1, int *p2);
```

Use this function to swap the values of two integers.

```
int main(int argc, char **argv){
    int x = 10;
    int y = 20;

    swap(/* what do I put here? */);

    printf("x: %d, y: %d\n", x, y);
}
```

Activity

Create a function `print_addr(int x)` whose sole purpose is to print the address of the integer `x` passed to it.

Create an integer variable in `main`, print out its address, and then pass that variable to `print_addr`.

Any observations about the behavior of this function?

C++

History of C++

- Began originally in 1979 with Bjarne Stroustrup's "C with Classes"
- C++ released in 1983 with most of the major features we know today.
- Design was strongly influenced by Simula, but Simula was too slow. Stroustrup wanted a fast, portable, language with object-oriented features. C had everything but OO.
- C++ is largely a superset of C. Until C++98, every C program was a valid C++ program. Still relatively easy to convert C to C++.
- Major driving philosophy: "only pay for what you use."
- C++ has many features. We will cover only the essential ones here.

How to use C++

- in file `helloworld.cpp`:

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello world!" << std::endl;
  return 0;
}
```

- compile code:

```
$ clang++ helloworld.cpp -o helloworld
```

- run program:

```
$ ./helloworld
```

- I strongly suggest that you explicitly specify a recent C++ standard:

```
$ clang++ -std=c++14 helloworld.cpp -o helloworld
```

C++ Classes

- C++ classes are similar in spirit to Java classes:

```
class Person {
private:
  string n;
public:
  Person(string name);
  void sayHello();
};

Person::Person(string name) : a(name) {}

void Person::sayHello() {
  cout << "Hello " << n << "!" << endl;
}
```

- This is called a *scope qualifier*. Without it, the compiler would not know that `Person` is the constructor for the `Person` class under separate compilation.

- Note that in C++, we conventionally put member function *definitions* after the class *declaration*. Purpose: separate compilation.
- Also note that C++ has a convenient string type that is much easier and safer to use than C-style strings (null-terminated char arrays).

C++ Classes

- As in Java, C++ classes can also inherit from a superclass.

```
class Person {
protected:
  string n;
public:
  Person(string name);
  virtual void sayHello();
};

…

class Pirate : public Person {
public:
  Pirate(string name);
  virtual void sayHello();
}

Pirate::Pirate(name) : Person(name) {}

void Pirate::sayHello() {
  cout << "Arr " << n << "!" <, endl;
}
```

- We changed this to `protected`. Why?
- We added the `virtual` keyword. Why?
- Class definitions end with a semicolon (ouch!)
- Person is a `public` superclass. What does this mean?
- We used weird constructor syntax here. Whaaaaaa?!!

Inheritance and Visibility Rules

- As in Java, C++ classes can also inherit from a superclass.

```
class Person {
protected:
  string n;
public:
  Person(string name);
  virtual void sayHello();
};
```

- `public`: instance variable or member function **is visible** to both inheriting classes and users of class.
- `protected`: instance variable or member function **is visible** to inheriting class **but not** users of class.
- `private`: instance variable or member function is **not visible** to either inheriting class or users of class.

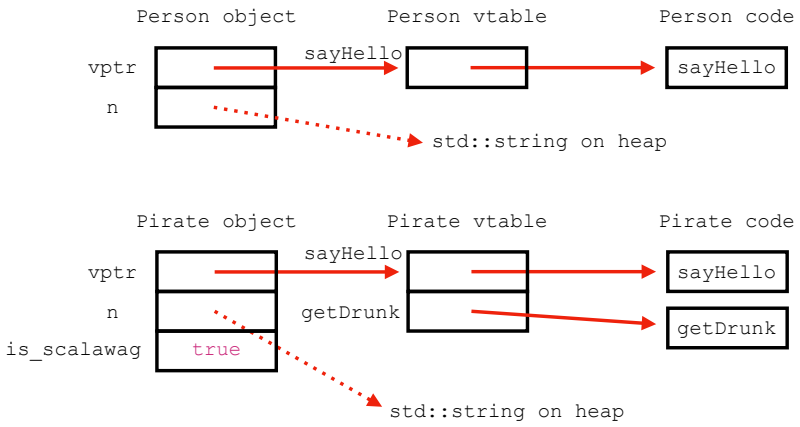
Virtual Dispatch

- In C++, you "only pay for what you use."

```
class Person {
protected:
  string n;
public:
  Person(string name);
  virtual void sayHello();
};
```

- Dynamic dispatch is "expensive" compared to static dispatch (two pointer dereferences and jump vs. direct jump)
- Therefore, the default is static dispatch; dynamic dispatch needs to be requested using the `virtual` keyword.
- This is often counterintuitive for Java programmers where dynamic dispatch is the default (as in Smalltalk).

Virtual Dispatch



- C++ virtual dispatch does *never searches* as in SmallTalk; vtable/instance variable offsets known at compile-time.

Next Class

- Templates
- Overloading
- Multiple inheritance
- Casting (eeew!)
- C++ lambdas
- Java!