Lecture 37: Encryption/Decryption, and Command Line Fu

Announcements and reminders

- Homework 9
  Due today! April 17 by 11 PM

- Project 3
  Due Wednesday April 24 by 11 PM

I'm going to be gone next week Monday afternoon - Thursday night.

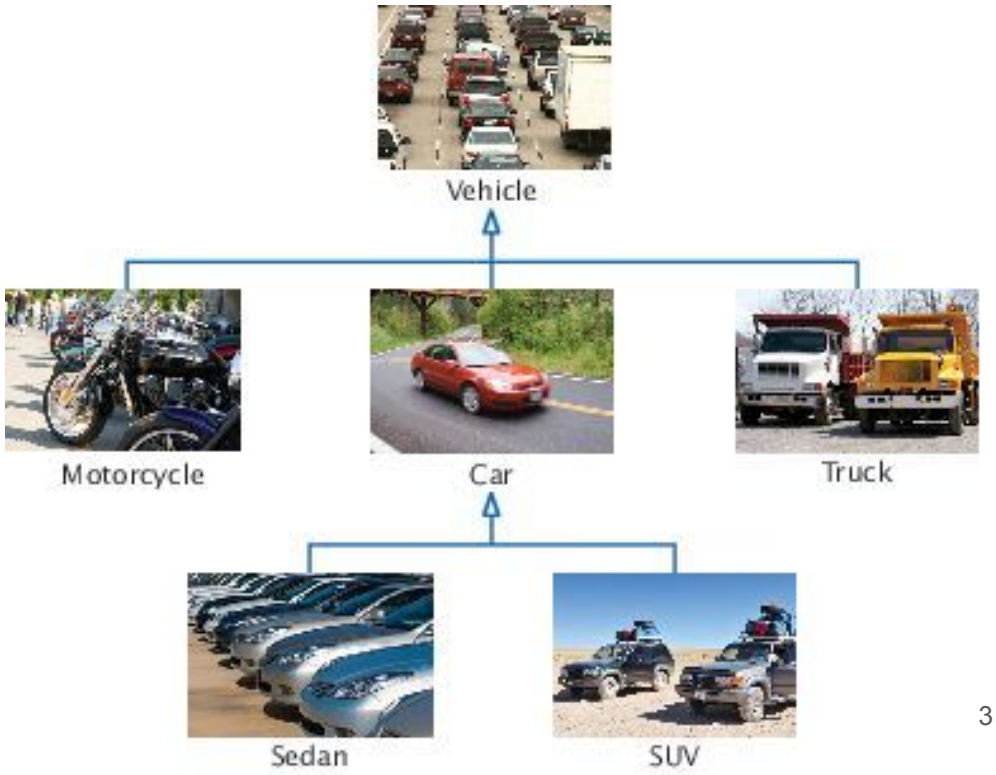
TA Karthik will cover lecture. It will be delightful!

→ **plan accordingly**

Last time on *Intro Computing…*

We saw **class inheritance!**

- ***Hierarchies*** showing an ***is a*** relationship
  - Ex: A car **is a** vehicle
  - Ex: An apartment **is a** household
- **Base class** vs **derived class**
  - Base = most general
  - Derived = more specific
- **Derived classes:**
  - New member fcns, data members
  - **Overriding** old member fcns



Command line

Depending on the operating system (OS) and C++ development system used, have different options for running a program:

- Select “Run” in the compilation environment
- Click on an icon somewhere
- Type the program name in a prompt in the command shell window
    - ***This*** is the method we’ve been using for compiling multiple files with our ***classes*****:**

```
g++ -o coolFileName.o -std=c++11 Player.cpp Team.cpp gameDriver.cpp
```

Command line

How to get the command shell window (terminal window) **from your OS**:

- **Windows:** type “cmd” in the Search box and click on **cmd.exe**
- **Mac:** Search for “terminal” and open it
- **Linux:** Search for “terminal” and open it

How to get the command shell window (terminal window) **in Cloud9**:

- Click on **Window** at the top toolbar,
- then click **New Terminal**

Command line arguments

No matter how you run your program, you can pass some information into the program via **command line arguments**

- These arguments are passed to the main function the same way you pass arguments into any old function
- Your ***execution*** of the program from the command line is actually ***calling*** the `main()` function!

It's a ***function…***

… so we can give that thing some ***input arguments***!

Command line arguments

For our program to process command line arguments, we must make a couple small changes to our **`main`** function:

```
int main(int argc, char* argv[])
{
    ... do stuff ...
}
```

`argc` = **arg**ument **c**ount. argc = 1 if the user typed nothing after the **program name** (1 arg)

`argv` = **arg**ument **v**ector. Not a ***real*** vector, but just a bunch of ***character pointers***

(behaves like a bunch of strings for the arguments you give)

Command line arguments -- some example values

greeting.cpp:

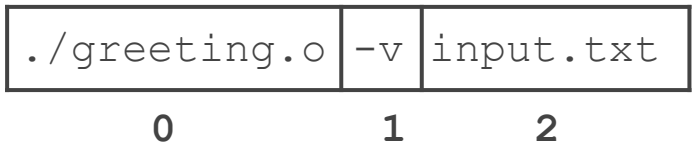
```
int main(int argc, char* argv[])
{
    ... do stuff ...
}
```

S'pose the user typed:

```
./greeting.o -v input.txt
```

argc = 3

argv =




```
argv[0] = "prog"
argv[1] = "-v"
argv[2] = "input.txt"
```

Command line arguments -- an example kind-of-mean program!

Example: Let's write a program that takes as **input from the command line** an optional argument to denote whether to use a **special greeting** (if present) or a **default greeting** (if not present).

Command line arguments -- an example kind-of-mean program!

Example: Let's write a program that takes as **input from the command line** an optional argument to denote whether to use a **special greeting** (if present) or a **default greeting** (if not present).

```
int main(int argc, char* argv[])
{
    // If argv[1] is "-g", then argv[2] is the greeting file name
    // --> Read and print that greeting
    // Otherwise,
    // --> Print a default greeting
}
```

Command line arguments -- an example kind-of-mean program!

```
int main(int argc, char* argv[]) {
    string arg = argv[1];      // coerce the character array into a string type
    if (arg=="-g") {
        ifstream infile;
        string filename = argv[2];
        string line;
        infile.open(filename);
        if (!infile.fail()) {
            getline(infile, line);
            cout << line << endl;     // special greeting!
        }
        infile.close();
    } else {
        cout << "Hey." << endl;  // default greeting
    }
    return 0;
}
```

Command line arguments -- an example kind-of-sneaky program!

Example: Let's write a program that encrypts/decrypts a file using a **caesar cypher**. Take as **input from the command line**:

- an input file name (to encrypt/decrypt)
- an output file name (for the encrypted/decrypted file)
- an optional flag `-d` to denote we should decrypt the file instead of encrypt

So, **our code will *not* prompt the user for file names!** We will pass them in as arguments:

```
./caesar.py
./caesar.py -d
```

Command line arguments -- an example kind-of-sneaky program!

Example: Let's write a program that encrypts/decrypts a file using a **caesar cypher**. Take as **input from the command line**:

- an input file name (to encrypt/decrypt)
- an output file name (for the encrypted/decrypted file)
- an optional flag `-d` to denote we should decrypt the file instead of encrypt

So, **our code will *not* prompt the user for file names!** We will pass them in as arguments:

To encrypt:

```
./caesar.o input.txt encrypted.txt
```

To decrypt:

```
./caesar.o -d encrypted.txt decrypted.txt
```

Command line arguments -- an example kind-of-sneaky program!

Fond memories of the Caesar cipher: Let's use an **encryption key** (shift) of 3.

Each character in our message is encrypted as that character + 3, circling around to *a* if we go past *z*. Let's assume only **lowercase** characters (and maybe spaces).

Plain text:	l	a	r	g	e		p	i	z	z	a

Encrypted text:	o	d	u	j	h		s	l	c	c	d

Let's code! *caesar.cpp*

What just happened?!

Command line arguments just happened!

- How we can supply arguments to our program directly
- No user-supplied values while the function is running
- Instead, give the values ***before*** the function executes
- **Great for automating code execution!**

Some encryption/decryption just happened too!

- Great for not getting murdered by your Roman friends!