EECS 280
DISCUSSION #6

Week of February 11

OUTLINE

- **Administrivia**
- You Can Make Variables Types Too!
- Look Who's Talking!

ADMINISTRIVIA

- Project 2
  - Grading done!
- Project 3
  - Due March 4 @ 11:59 PM
  - You are done, right?

OUTLINE

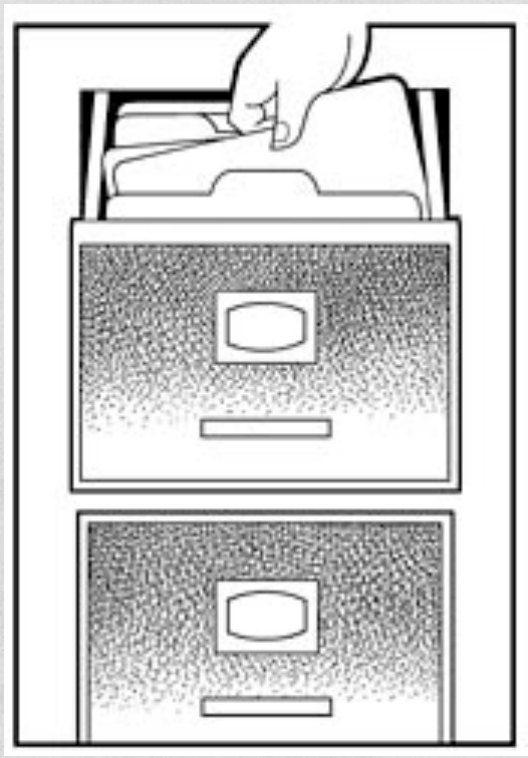
- Administrivia
- **You Can Make Variables Types Too!**
  - Why?
  - Structures
  - Enumerations
  - Composability
- Look Who's Talking!

GASP!

- An ugly secret of computer science: all programs can be summarized as the following [possibly repeating/intertwining] sequence
  - a) Get data
  - b) Manipulate data
  - c) Output data
- Whatever you can do to most safely, efficiently, and cost-effectively get from (a) to (c) makes you a good programmer
- Custom variable types can make data manipulation easier, faster, and more maintainable over time

MOTIVATION: STRUCTURES

- Let's keep track of a students, what do we need for <u>each</u> student?
  - A picture
  - Name (and other basic info)
  - A list of grades
  - Disciplinary actions



STRUCTURES: DEFINITION

Define the fields (member elements)

```
struct student_info
{
  string name;
  string phone;
  char final_grade;
};
```

STRUCTURES: USAGE

Now that we have a definition of our new structure, we just declare a new variable (or many) of that type:

```
student_info a;

a.name = “nate”;

a.phone = “734-555-1212”;

a.final_grade = ‘B’;


student_info b = {“jenny”, “867-5309”, ‘C’};
```

ENUMERATION

- Definition (Oxford American):
  - Mention (a number of things) one by one
- Computer Science view (Wikipedia):
  - Model an attribute that has a specific number of options
  - ex: card suits

ENUM: DEFINITION

Define the possible values

```
enum card_suit
{
  hearts,
  diamonds,
  clubs,
  spades
};
```

ENUM: USAGE

Now that we have a definition of our new structure, we just declare a new variable of that type:

```
card_suit trump;

trump = hearts;

if ( trump == spades )

...
```

COMPOSABILITY

- Custom variable types can contain other custom variable types
- Challenge!
  - Develop custom variable types to represent a deck of playing cards

COMPOSABILITY EXAMPLE

```
enum card_suit
{
  hearts, diamonds,
  clubs, spades
};

enum card_value
{
  two, three, four, five, six, seven, eight,
  nine, ten, jack, queen, king, ace
};
```

COMPOSABILITY EXAMPLE

```
struct card_type
{
  card_suit suit;
  card_value value;
};

struct deck_of_cards
{
  int current_card;
  card_type cards[52];
};
```

OUTLINE

- Administrivia
- You Can Make Variables Types Too!
- **Look Who's Talking!**
  - Telling your program what to do
  - Reading, 'riting, 'rithmetic

DO AS I SAY!

- Consider a call to the compiler:
  - `g++ -Werror -Wall -m32 p3.cpp dice.cpp -o p3`
- Everything after "g++" is an argument to the compiler
- That's right - programs can take arguments just like functions!

WHAT DID YOU SAY?

- Access program arguments via arguments to the "main" function:
  - `int main(int argc, char *argv[])`
  - `argc = number of arguments`
  - `argv = c-style strings, representing the actual argument values`

ARGUMENTS: EXAMPLE

```
g++ -Werror -Wall -m32 p3.cpp dice.cpp -o p3

argc = 8

argv[0] = “g++”

argv[1] = “-Werror”

argv[2] = “-Wall”

...

argv[7] = “p3”
```

STREAMS

- Transfer data from one point to another
  - cin: console -> program
  - cout: program -> console
  - fstream: program <=> file

STEP 1: LIBRARIES

- Input/Output Streams
  - #include <iostream>
  - cin and cout
- File Streams
  - #include <fstream>
  - ifstream, ofstream

STEP 2: OPEN THE STREAM

- cin and cout are "opened" automatically
- Input files:
  - `ifstream input_file;`
  - `input_file.open( "filename.ext" );`
- Output files:
  - `ofstream output_file;`
  - `output_file.open( "filename.ext" );`
- Check for failure:
  - `if ( my_file.fail() ) { do_something(); }`

STEP 3: OPERATORS GALORE!

- Many functions/operators useful for manipulating streams
- Output
  - insertion: <<
- Input
  - extraction: >>
  - getline
  - get

OP: INSERTION

- The "<<" (insertion) operator places data on an output stream (cout, ofstream)
- Output is buffered till "flushed"
  - Done automatically via "endl"
- Useful functions
  - setw (#include <iomanip>)

OP: EXTRACTION

- The “>>” (extraction) operator pulls type-specific data from an input stream (cin, ifstream)
- Ignores white-space (space, tab, new line)
- Errors in data types can be detected via “fail” and cleared via “clear”
  - Buffer is reset to beginning of error

EXTRACTION EXAMPLES

- Input: “3.14159 is about pi” (sans quotes)
- int i, string s, char c, float f
  - cin >> f >> s >> c — f = 3.14159, s=”is”, c=’a’
  - cin >> i >> c >> s — i = 3, c=’.’, s=”14159”
  - cin >> i >> s >> c — i = 3, s=”.14159”, c=’i’

FUNCTION: GETLINE

- Captures data from an input stream (cin, ifstream) till the end of the current line
  - stream.getline => char *
  - getline(stream, string) => c++ string

FUNCTION: GET

- Grabs a single character from an input stream (cin, ifstream)
  - char c = stream.get();
- Does NOT ignore white space

STEP 4: CLOSE THE STREAM

- cin and cout are automatically closed
- File streams must be closed manually
  - `my_file.close();`

FINAL THOUGHTS

HAPPY V-DAY :)