CSCI 200: Foundational Programming Concepts & Design Lecture 16

Input Paradigms & Validation

Output Formatting

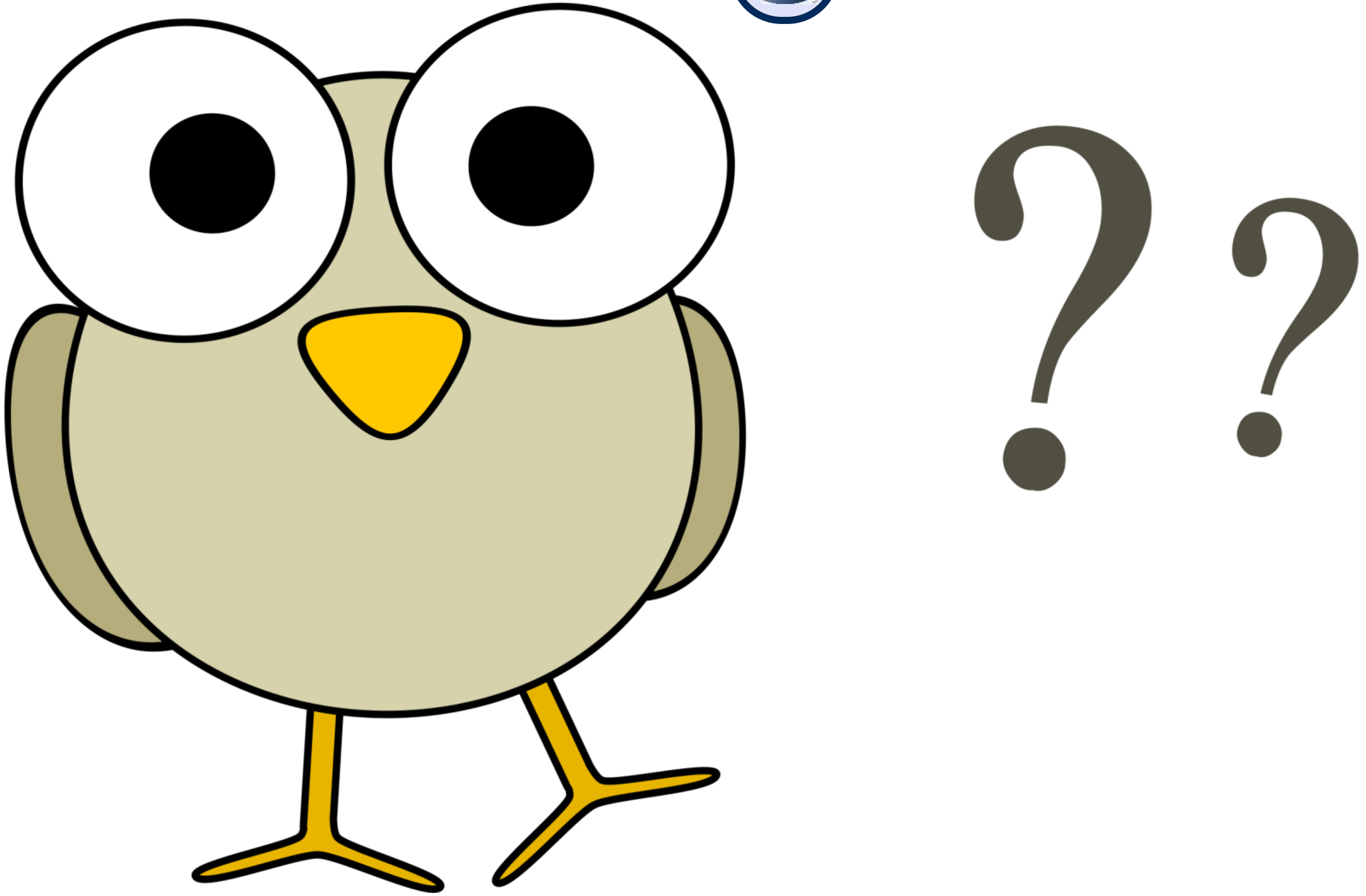
Log into Canvas & iClicker

Download Lecture 16 starter code

Previously in CSCI 200

- Reading/Writing to a file: 6 steps to read/write
    1. Include header
    2. Declare file stream
    3. Open file
    4. Check for error opening
    5. Read/Write data
    6. Close file
- Functions associated with file streams:
    - open() / close()
    - fail() / is_open()
    - eof() / get()

Questions?



Learning Outcomes For Today

- Define REPL and perform read operations conforming to common input paradigms.
- Create a program that validates user input and removes the need for a cooperative smart user.
- Create a program with formatted output.

On Tap For Today

- Reading Paradigms
- Stream Errors
- Formatting
- Practice

On Tap For Today

- Reading Paradigms
- Stream Errors
- Formatting

- Practice

Reading Files Boilerplate

```cpp
#include <fstream>
#include <iostream>
using namespace std;
int main() {
  ifstream myDataIn( "FILENAME" );
  if( myDataIn.fail() ) {
    cerr << "Could not open \"FILENAME\"" << endl;
    return -1;
  }
  char x; // or int x, double x, etc.
  while( !myDataIn.eof() ) {
    myDataIn >> x;
    // do marvelous things and print results
  }
  myDataIn.close();
  return 0;
}
```

Reading Files Boilerplate

```
#include <fstream>
#include <iostream>
using namespace std;
int main() {
  ifstream myDataIn( "FILENAME" );
  if( myDataIn.fail() ) {
    cerr << "Could not open \"FILENAME\"" << endl;
    return -1;
  }
  char x; // or int x, double x, etc.
  while( !myDataIn.eof() ) {
    myDataIn >> x;
    // do marvelous things and print results
  }
  myDataIn.close();
  return 0;
}
```

Read

Evaluate

Print

Loop

Reading Paradigms

- Dependent on how data in our file is formatted

Reading Paradigms

- Dependent on how data in our file is formatted
  - First Line of file = number of lines (records)
  - Counter-controlled loop

datafile.txt

```
4
1 3
2 4
5 6
7 8
```

```
ifstream fileIn( "datafile.txt" );
int numLines;
fileIn >> numLines;
for( int i = 0; i < numLines; i++ ) {
  int x, y;
  fileIn >> x >> y;
  // do magic
}
```

Reading Paradigms

- Dependent on how data in our file is formatted
  - Last Line of file = special value to indicate data end
  - Sentinal-controlled loop

datafile.txt

```
4
1
2
5
-9999
```

```
const int SENTINAL_VALUE = -9999;
ifstream fileIn( "datafile.txt" );
int x;
while( true ) {
  fileIn >> x;
  if( x == SENTINAL_VALUE ) break;
  // do magic
}
```

Reading Paradigms

- Dependent on how data in our file is formatted
  - No knowledge within file
  - End-of-data loop

datafile.txt

```
4
1
2
5
3
```

```
ifstream fileIn( "datafile.txt" );
while( !fileIn.eof() ) {
  int x;
  fileIn >> x;
  // do magic
}
```

On Tap For Today

- Reading Paradigms
- Stream Errors
- Formatting

- Practice

Input Error

A. `int i, j;`
   `cin >> i >> j;`

B. `double x, y;`
   `cin >> x >> y;`

C. `char c1, c2;`
   `cin >> c1 >> c2;`

Data entered

```
1.2.3
```

- What is the value of each variable?

Input Errors Occur When..

- Input != variable type
- Stream variable placed in error state
- All future inputs ignored

- To Do
  1. Test whether reading variable is in error state
  2. If yes, print to cerr and exit

Example

```
int main() {
  int x, y;
  cin >> x >> y;
  if( cin.fail() ) {
    cerr << "error encountered from read" << endl;
    return -1;
  }
  // do other stuff
}
```

Input Errors Occur When..

- Input != variable type
- Stream variable placed in error state
- All future inputs ignored

- To Do
  1. Test whether reading variable is in error state
  2. If yes, print to cerr and ~~exit~~ goto end of line and try again

Example

```
int main() {
  int x, y;
  while( true ) { // loop until we get good data
    cin >> x >> y;

    if( !cin.fail() ) break;  // we succeeded, break out of loop

    cerr << "error encountered from read" << endl;
    cin.clear(); // clear error

    char badChar; // clear out rest of input line
    do { badChar = cin.get(); } while( badChar != '\n' );
    cout << "Enter two integers: ";
  }
  // do other stuff
}
```

Accepting Only Valid Values

- May receive data of proper data type, but may be wrong value

```
int main() {
  while( true ) { // loop until we get good data
    char userValue;
    cout << "Enter q to quit: ";
    cin >> userValue;
    if( userValue == 'q' ) break;

    cout << "Invalid value." << endl;
  }
}
```

On Tap For Today

- Reading Paradigms
- Stream Errors
- Formatting

- Practice

Simple Formatting

- Special “escape characters” for output

  '\n' → new line

  '\t' → tab

  '\\' → backslash aka whack

  "\"" → quotation mark (whack double quote)

  '\'' → apostrophe (whack single quote)

  ' ' → space

`'\n'` or `endl` ?

- Both insert newline

- **`endl`** also flushes the stream

```
cout << "print now" << flush;
// ... later on ...
cout << " add to prior output" << endl;
```

Output Manipulators

- Include the iomanip library

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

- Manipulators modify output format
- Can modify output to any destination (standard out or file out)
  - Each destination has its own formatting
  - Need to specify formatting per destination

iomanip

- Examples

```
// floating point display
cout << fixed;                // use decimal notation
cout << scientific;           // use scientific notation
cout << setprecision( 3 ); // 3 positions of precision

// alignment
cout << setw( 10 );           // set width of 10 to display
cout << left;                 // left align in column
cout << right;                // right align in column
cout << setfill('-');         // fill allocated space with
```

On Tap For Today

- Reading Paradigms
- Stream Errors
- Formatting

- Practice

Pointers Quiz

- Make Canvas Full Screen
  - Put everything else away
- Access Code:
- 8 Minutes

To Do For Next Time

- Complete the readings and videos on `vector` and `string` APIs
- Have a good weekend!