CS180 - Error Handling & Large Projects

Note Title 2/4/2011

Announcements

- Lab due tomorrow (Sat.) by 11:59pm
- HW2 due Tuesday by 11:59pm
- For the homeworks, email to grader account CSCI180HW2011@gmail.com

Raising Exceptions (Transition Guide)

In Python:

```
def sqrt(number):
  if number < 0:
    raise ValueError('number is negative')
```

In C++:

```
double sqrt(double number) {
  if (number < 0)
    throw domain_error("number is negative");
```

(assuming we include <stdexcept>)

Catching Exceptions

Generic Example:

```
try {
  // any sequence of commands, possibly nested
} catch (domain_error& e) {
  // what should be done in case of this error
} catch (out_of_range& e) {
  // what should be done in case of this error
} catch (exception& e) {
  // catch other types of errors derived from exception class
} catch (...) {
  // catch any other objects that are thrown
}
```

So to catch:

```
double sqrt(double number) {
  if (number < 0)
    throw domain_error("number is negative");
```

```
int main() {
    double n;
    cin >> n;
    try {
        cout << "The square root is " << double(n) << endl;
    }
    catch (domain_error & e) {
        cout << e << endl << "Please try again" << endl;
    }
}
```

Other errors

By default, cin doesn't raise errors when something goes wrong. Instead, it sets flags.

Use cin.bad(), cin.fail(), etc., to detect.

Can get a bit long - see p.27 of transition guide for more details.

Example →

```
number = 0;
while (number < 1 || number > 10) {
  cout << "Enter a number from 1 to 10: ";
  cin >> number;
  if (cin.fail( )) {
    cout << "That is not a valid integer." << endl;
    cin.clear( );                                        // clear the failed state
    cin.ignore(std::numeric_limits<int>::max( ), '\n');  // remove errant characters from line
  } else if (cin.eof( )) {
    cout << "Reached the end of the input stream" << endl;
    cout << "We will choose for you." << endl;
    number = 7;
  } else if (cin.bad( )) {
    cout << "The input stream had fatal failure" << endl;
    cout << "We will choose for you." << endl;
    number = 7;
  } else if (number < 1 || number > 10) {
    cout << "Your number must be from 1 to 10" << endl;
  }
}
```

Files & error handling

(more similar to cin, unfortunately)

```
void openFileReadRobust(ifstream& source) {
  source.close( );                          // disregard any previous usage of the stream
  while (!source.is_open( )) {
    string filename;
    cout << "What is the filename? ";
    getline(cin, filename);
    source.open(filename.c_str( ));
    if (!source.is_open( ))
        cout << "Sorry. Unable to open file " << filename << endl;
  }
}
```

useful in this class

Larger Projects

In larger projects, we often separate into multiple files.

- Easier version control
- Allows division of files in a group

.h files

Header files are used to declare the interface of a class or set of functions, but don't actually define them:

(no actual code here)

gcd.h

```
#ifndef GCD_H
#define GCD_H
int gcd(int u, int v); // forward declaration
#endif
```

idea - user can open .h file to get info on how to use the class (& ignore the details of it works)

Point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>      // need ostream definition for operator<< signature

class Point {
private:
  double _x;
  double _y;

public:
  Point(double initialX=0.0, double initialY=0.0);
  double getX( ) const { return _x; }                    // in-lined function body
  void setX(double val) { _x = val; }                    // in-lined function body
  double getY( ) const { return _y; }                    // in-lined function body
  void setY(double val) { _y = val; }                    // in-lined function body
  void scale(double factor);
  double distance(Point other) const;
  void normalize( );
  Point operator+(Point other) const;
  Point operator*(double factor) const;
  double operator*(Point other) const;
};   // end of Point class

// Free-standing operator definitions, outside the formal Point class definition
Point operator*(double factor, Point p);
std::ostream& operator<<(std::ostream& out, Point p);
#endif
```

Other files

We then usually have 2 kinds of cpp files:

- One to declare functions or classes:

must include the related .h file

gcd.cpp

```
#include "gcd.h"

int gcd(int u, int v) {
  /* We will use Euclid's algorithm
     for computing the GCD */
  int r;
  while (v != 0) {
    r = u % v;    // compute remainder
    u = v;
    v = r;
  }
  return u;
}
```

- The other to have the main program:

gcdTest.cpp

```
#include "gcd.h"
#include <iostream>
using namespace std;

int main( ) {
  int a, b;
  cout << "First value: ";
  cin >> a;
  cout << "Second value: ";
  cin >> b;
  cout << "gcd: " << gcd(a,b) << endl;
  return 0;
}
```

don't include gcd.cpp

Part of Point.cpp :

```
#include "Point.h"
#include <iostream>                // for use of ostream
#include <cmath>                   // for sqrt definition
using namespace std;               // allows us to avoid qualified std::ostream syntax

Point::Point(double initialX, double initialY) : _x(initialX), _y(initialY) { }

void Point::scale(double factor) {
  _x *= factor;
  _y *= factor;
}

double Point::distance(Point other) const {
  double dx = _x - other._x;
  double dy = _y - other._y;
  return sqrt(dx * dx + dy * dy);          // sqrt imported from cmath library
}

void Point::normalize( ) {
  double mag = distance( Point( ) );       // measure distance to the origin
  if (mag > 0)
    scale(1/mag);
}

Point Point::operator+(Point other) const {
  return Point(_x + other._x, _y + other._y);
}
```

scope to Point.h

std::cin

Alternative, you could use 1 file:

```
#include <iostream>
:

class Point {
  private:
    ....
  public:
    ....
}

int main() {
  test
}
```

(but becomes difficult when it gets long)

Compiling & Linking

- Complication: main can't run without functions or classes!

We have to compile these in the correct order

When gcd was all 1 file, was:

```
g++ -o gcd gcd.cpp
```

Now:

```
g++ -o gcd gcd.cpp gcdTest.cpp
```

these 2 things get compiled

name executable gcd

Also:

g++ gcd.cpp → outputs .o file

g++ gcdTest.cpp (no main, so no executable)

then

```
g++ -o gcd gcd.o gcdTest.o
```

---

g++ Point.cpp

g++ PointTest.cpp

} → a.out

Alternatively:

Makefiles are used to automate this.

I'll generally provide a makefile.

If you use the names I suggest, you'll just need to type "make".

(post template on schedule page)

end if
not defined

#ifndef and #define and #endif

Use these at begining & end of all our files.

if not defined

Loads file if it hasn't already been done.

Debugging

output everything!

output variables

output "here" statements &
figure out where the problem is