

C++ Kitchen Sink

Lecture 14



Outline

- Exceptions
- Iterators
- Random numbers
- Casting
- Enumerations
- Pairs
- The Big Three

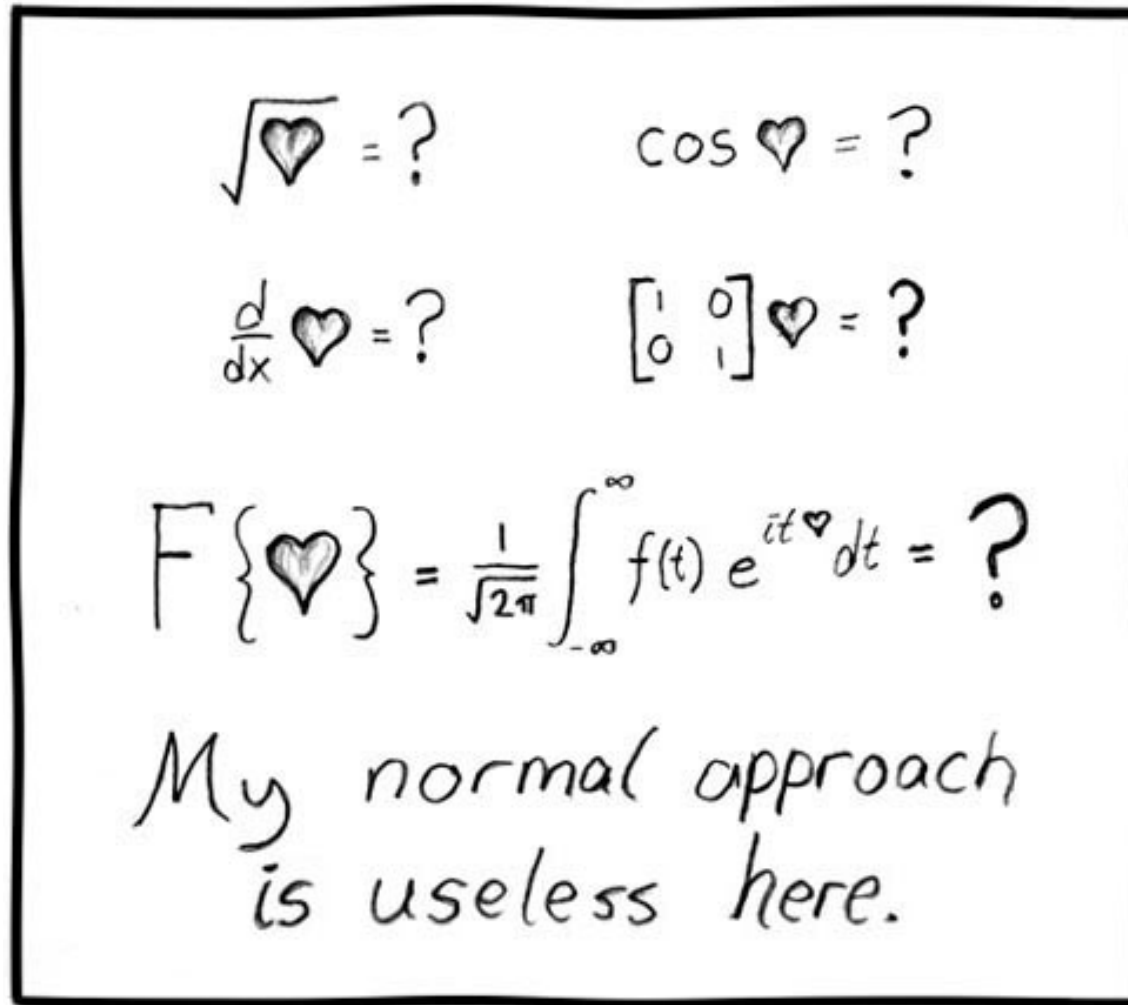


Error Handling

- It is often easier to write a program by first assuming that nothing incorrect will happen
- Once it works correctly for the expected cases, add code to handle the exceptional cases
- *Exception* handling is commonly used to handle error situations
 - Once an error is handled, it is no longer an error



Motivational XKCD



Exceptions

- In C++, exception handling proceeds by:
 - Some library software or your code signals that something unusual has happened
 - This is called *throwing an exception*
 - At some other place in your program you place the code that deals with the exceptional case
 - This is called *handling or catching* the exception



Example

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

```
int main()
{
    double a, b;

    cin >> a >> b;
    cout << ( a / b )
         << endl;

    return 0;
}
```

```
./a
3
5
0.6
./a
3
0
inf
```



Try-Throw-Catch

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

int main()
{
    double a, b;
    cin >> a >> b;

    try
    {
        if ( b == 0 )
            throw a;

        cout << ( a / b )
             << endl;
    }
    catch (double a)
    {
        cout << "Can't divide "
             << a << " by zero"
             << endl;
    }

    return 0;
}
```

```
./a
3
5
0.6
./a
3
0
Can't divide 3 by zero
```



The Basics

- In C++ you can throw any type
 - Commonly a class that carries useful information to catch
- Code within the try block is suspected to throw an exception
 - Similar to if-else, but with messaging
- There can be multiple catch blocks with different types



Advanced Example

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

class DivideException
{
public:
    DivideException(double val):
        val( val ) {}
    double getVal() { return val; }

private:
    double val;
};

double div(double a, double b)
throw (DivideException)
{
    if ( b == 0 )
        throw DivideException( a );

    return ( a / b );
}
```

```
int main()
{
    double a, b;

    cin >> a >> b;

    try
    {
        cout << div( a, b ) << endl;
    }
    catch (DivideException e)
    {
        cout << "Can't divide "
             << e.getVal()
             << " by zero" << endl;
    }

    return 0;
}
```

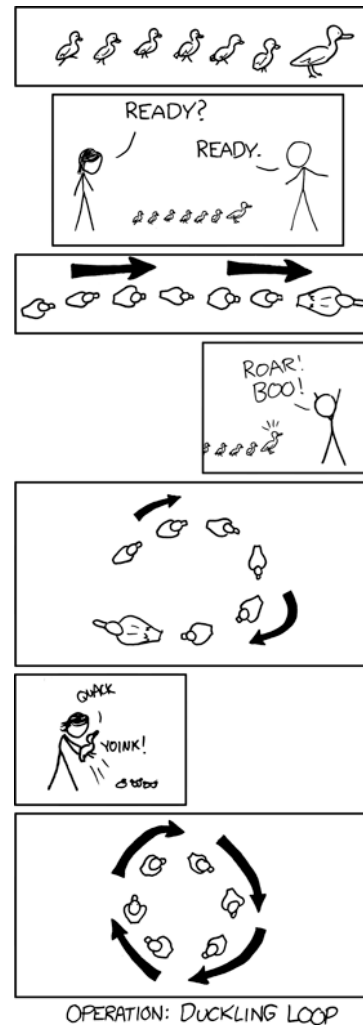


Iterators

- An iterator is a generalization of a pointer
 - Typically implemented using a pointer, but saves you from the details and provides a uniform interface to numerous collection classes
- Think of an iterator like a pointer
 - Has many similar operators overloaded



Motivational XKCD



Common Usage

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

int main()
{
    vector<int> v;
    v.push_back( 1 );
    v.push_back( 2 );
    v.push_back( 3 );

    for ( vector<int>::iterator it=v.begin();
          it!=v.end();
          it++ )
    {
        cout << *it << endl;
    }

    return 0;
}
```

1
2
3



Random Number Generator

- With most computers it is difficult to devise a method by which to produce truly random numbers
 - These are useful in many situations (e.g. games, statistic simulation, AI, ...)
- A random number generator (RNG) is an algorithm designed to generate a sequence of numbers or symbols that appear random



Motivational XKCD

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```



The Basic Pattern

- With most RNGs, there is the concept of a *seed* – a value that is used to produce a distinct sequence
 - If you provide the same seed, you should always get the same sequence
- So...
 - Seed the RNG
 - Ask for as many random values as you need
 - May need computation to get in useful bounds



Example

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

int main()
{
    srand( 42 ); // seed the rng
    for ( int i=0; i<5; i++ )
    {
        // rand() = [0, RAND_MAX]
        cout << ( rand() % 100 )
              << endl;
    }

    return 0;
}
```

```
94
23
9
43
26
```



Casting

- Whenever you need to convert between data types, a form of type *casting* is involved
- There are many issues, including pointers, inheritance, const




Motivational XKCD (a stretch)

GUIDE TO CONVERTING TO METRIC

TEMPERATURE


60°C	EARTH'S HOTTEST
45°C	DUBAI HEAT WAVE
40°C	SOUTHERN US HEAT WAVE
35°C	NORTHERN US HEAT WAVE
30°C	BEACH WEATHER
25°C	WARM ROOM
20°C	ROOM TEMPERATURE
10°C	JACKET WEATHER
0°C	SNOW!
-5°C	COLD DAY (BOSTON)
-10°C	COLD DAY (MOSCOW)
-20°C	FUCKFUCKFUCKCOLD
-30°C	FUUUUUUUUUCK!
-40°C	SPT GOES "CLINK"



THE KEY TO CONVERTING TO METRIC IS ESTABLISHING NEW REFERENCE POINTS. WHEN YOU HEAR "26°C," INSTEAD OF THINKING "THAT'S 79°F" YOU SHOULD THINK, "THAT'S WARMER THAN A HOUSE BUT COOL FOR SWIMMING." HERE ARE SOME HELPFUL TABLES OF REFERENCE POINTS:

LENGTH

1 cm	WIDTH OF MICROSD CARD
3 cm	LENGTH OF SD CARD
12 cm	CD DIAMETER
14 cm	PENIS
15 cm	BIC PEN
80 cm	DOORWAY WIDTH
1 m	LIGHTSABER BLADE
170 cm	SUMMER GLAU
200 cm	DARTH VADER
2.5m	CEILING
5m	CAR-LENGTH
16m ^{4m}	HUMAN TOWER OF SERENITY CREW



SPEED

kph	m/s	
5	1.5	WALKING
13	3.5	JOGGING
25	7	SPRINTING
35	10	FASTEST HUMAN
45	13	HOUSECAT
55	15	RABBIT
75	20	RAPTOR
100	25	SLOW HIGHWAY
110	30	INTERSTATE (65 MPH)
120	35	SPEED YOU ACTUALLY GO WHEN IT SAYS "65"
140	40	RAPTOR ON HOVERBOARD

VOLUME

3 mL	BLOOD IN A FIELDMOUSE
5 mL	TEASPOON
30 mL	NASAL PASSAGES
40 mL	SHOT GLASS
350 mL	SODA CAN
500 mL	WATER BOTTLE
3 L	TWO-LITER BOTTLE
5 L	BLOOD IN HUMAN MALE
30 L	MILK CRATE
55 L	SUMMER GLAU
65 L	DENNIS KUCINICH
75 L	RON PAUL
200 L	FRIDGE

SO, WHEN IT'S BLOCKED, THE MUCUS IN YOUR NOSE COULD ABOUT FILL A SHOT GLASS.

RELATED: I'VE INVENTED THE WORST MIXED DRINK EVER.

$55 + 65 + 75 < 200$



MASS

3g	PEANUT M&M
100g	CELL PHONE
500g	BOTTLED WATER
1 kg	ULTRAPORTABLE LAPTOP
2 kg	LIGHT-MEDIUM LAPTOP
3 kg	HEAVY LAPTOP
5 kg	LCD MONITOR
15 kg	CRT MONITOR
4 kg	CAT
4.1 kg	CAT (WITH CAPTION)
60 kg	LADY
70 kg	DUDE
150 kg	SHAQ
200 kg	YOUR MOM
220 kg	YOUR MOM (INCL. CHEAP JEWELRY)
223 kg	YOUR MOM (ALSO INCL. MAKEUP)




Example

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

class Base
{
public:
    void foo()
    {
        cout << "Base" << endl;
    }
};

class Derived: public Base
{
public:
    void foo()
    {
        cout << "Derived" << endl;
    }
};
```

```
int main()
{
    Base* b = new Base();
    Base* d = new Derived();

    b->foo();
    d->foo();
    ((Derived*) d)->foo();

    return 0;
}
```

Base
Base
Derived

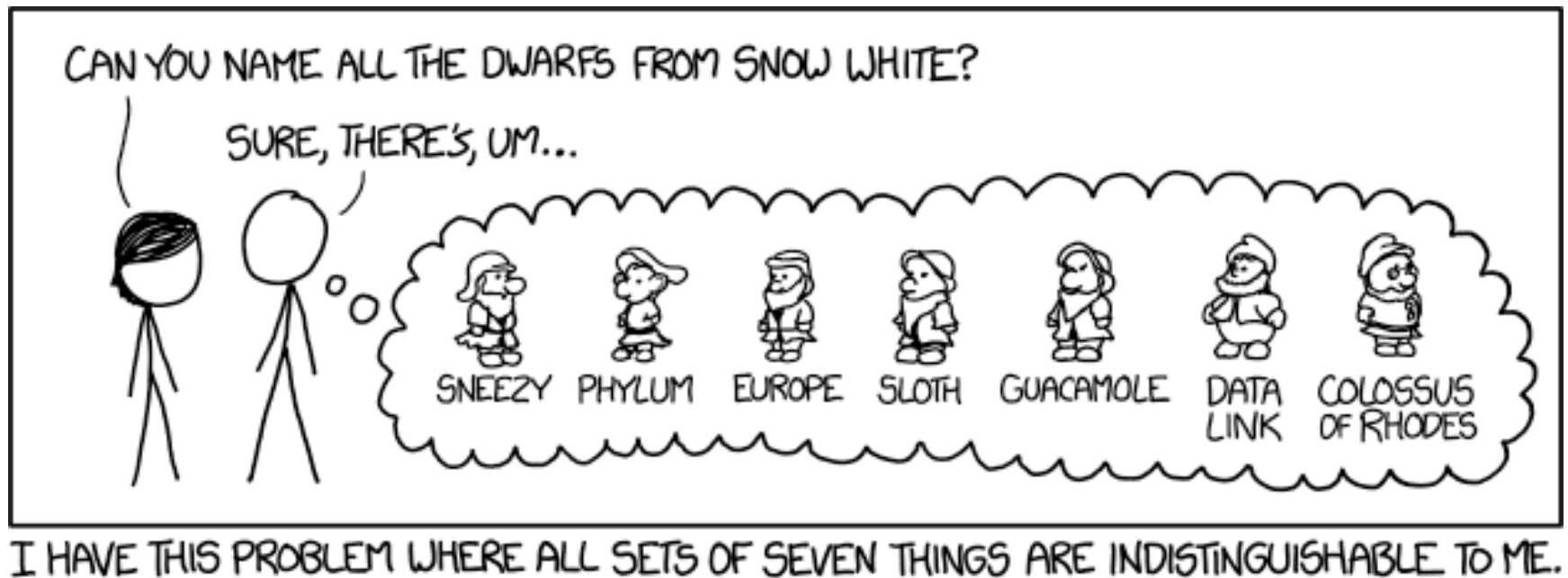


Enumerations

- A distinct type whose value is restricted to one of several explicitly named constants
- The values of the constants are values of an integral type
- Useful to let the compiler keep you from making mistakes when you know the range of values



Motivational XKCD



Example

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

enum suit
{
    hearts, diamonds, spades, clubs
};

int main()
{
    suit a = hearts;
    suit b = spades;

    cout << ( a == b ) << endl;

    return 0;
}
```

0

Pairs

- It's often useful to bundle two variables together, and so there is a C++ `pair` class
- Pairs are templated, and generically referred to by the **first** or **second** element



Motivational XKCD



Example

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

int main()
{
    pair<char, int> a =
        make_pair<char, int>( 'a', 1 );

    cout << a.first
         << a.second
         << endl;

    return 0;
}
```



a1



Deep vs. Shallow Copies

- When assigning objects, a *shallow* copy is performed
 - Values are copied between instances
- This becomes problematic when dealing with dynamically allocated memory
 - All that is copied is a memory address, and so now two objects point to the same memory
- **Rule of the Big Three.** If you have any dynamically allocated storage in a class, you must provide:
 1. A destructor
 2. A copy constructor
 3. An overloaded assignment operator



Motivational XKCD



Shallow Example

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

class Foo
{
public:
    int bar;
};

int main()
{
    Foo a, b;
    a.bar = 5;
    b = a;
    a.bar = 7;

    cout << a.bar << " "
         << b.bar << endl;

    return 0;
}
```

7 5



Pointer Problem

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

class Foo
{
public:
    int* bar;
};

int main()
{
    Foo a, b;
    a.bar = new int(5);
    b = a;

    *(a.bar) = 7;

    cout << *(a.bar) << " "
         << *(b.bar) << endl;

    return 0;
}
```

7 7



The Big Three

```
class Foo
{
public:
    Foo() { bar = new int(0); }
    Foo(int x) { bar = new int(x); }

    Foo(const Foo& other)
    {
        bar = new int( *(other.bar) );
    }

    ~Foo() { delete bar; }

    Foo& operator =(const Foo& other)
    {
        *bar = *(other.bar);
        return *this;
    }

    int* bar;
};
```

```
int main()
{
    Foo a(5), b;
    Foo c(a);
    b = a;

    *(a.bar) = 7;

    cout << *(a.bar) << " "
         << *(b.bar) << " "
         << *(c.bar) << endl;

    return 0;
}
```

7 5 5



Wrap Up

- There are many aspects of C++ that we did not cover in depth
- I hope you continue to explore programming in C++ and other languages
 - Much of what we covered here has its parallels in other languages



End-of-Semester XKCD

