A Rose by any other name…C or Java?

- Why do we use Java in our courses (royal we?)
  - Object oriented
  - Large collection of libraries
  - Safe for advanced programming and beginners
  - Harder to shoot ourselves in the foot
- Why don't we use C++ (or C)?
  - Standard libraries weak or non-existant (comparatively)
  - Easy to make mistakes when beginning
  - No GUIs, complicated compilation model

Why do we learn other languages?

- Perl, Python, PHP, mySQL, C, C++, Java, Scheme, ML, …
  - Can we do something different in one language?
    - Depends on what different means.
    - In theory: no; in practice: yes
  - What languages do you know? All of them.
  - In what languages are you fluent? None of them
- In later courses why do we use C or C++?
  - Closer to the machine, we want to understand the machine at many levels, from the abstract to the ridiculous
    - Or at all levels of hardware and software
  - Some problems are better suited to one langauge
    - What about writing an operating system? Linux?

C++ on three slides

- Classes are similar to Java, compilation model is different
  - Classes have public and private *sections/areas*
  - Typically declaration in .h file and implementation in .cpp
    - Separate interface from actual implementation
    - Good in theory, hard to get right in practice
  - One .cpp file compiles to one .o file
    - To create an executable, we *link* .o files with libraries
    - Hopefully someone else takes care of the details (Makefile)
- We #include rather than import, this is a preprocessing step
  - Literally sucks in an entire header file, can take a while for standard libraries like iostream, string, etc.
  - No abbreviation similar to java.util.*;

C++ on a second slide

- We don't have to call new to create objects, they can be created "on the stack"
  - Using new creates memory "on the heap"
  - In C++ we need to do our own garbage collection, or avoid and run out of memory (is this an issue?)
- `vector` similar to `ArrayList`, pointers are similar to arrays
  - Unfortunately, C/C++ equate array with memory allocation
  - To access via a pointer, we don't use . we use ->
- Streams are used for IO, iterators are used to access begin/end of collection
  - ifstream, cout correspond to Readers and System.out

How do we read a file? (SearchDemo)

```
TreeSet<String> unique = new TreeSet<String>();
int total = 0;
while (s.hasNext()){
   String str = s.next();
   total++;
   unique.add(str.toLowerCase());
}
myWordsAsList = new ArrayList(set);
```

```
string word;
set<string> unique;
int total = 0;
while (input >> word){
   transform(word.begin(), word.end(),
             word.begin(),makelower); // ml NOT standard
   unique.insert(word);
   total++;
}
myWords = vector<string>(unique.begin(), unique.end());
```


Shafi Goldwasser

- RCS professor of computer science at MIT
  - Co-inventor of zero-knowledge proof protocols

How do you convince someone that you know something without revealing "something"

- ACM Grace Murray Hopper award and Godel prize in Theoretical Computer Science (twice)

Work on what you like, what feels right, I now of no other way to end up doing creative work

Toward an Understanding of C++

- Traditional first program, doesn't convey power of computing but it illustrates basic components of a simple program

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

// traditional first program

int main()
{
   cout << "Hello world" << endl;
   return 0;
}
```

- This program must be edited/typed, compiled, linked and executed.
- Other languages don't use compile/link phase, examples?

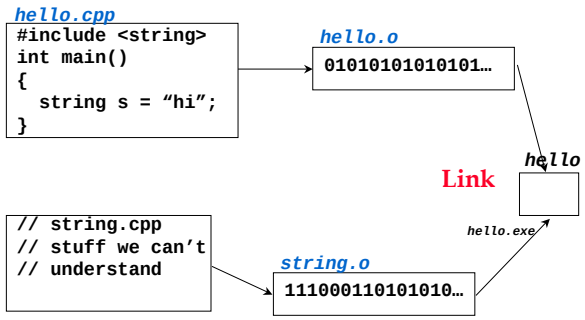
What's a namespace?

- In "standard" C++, objects and types are classified as to what namespace they're in. Hierarchy is good.

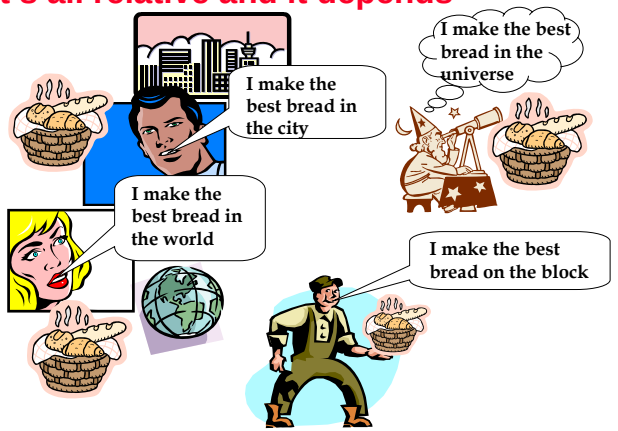
```
#include <iostream>

// traditional first program

int main()
{
   std::cout << "Hello world" << std::endl;
   return 0;
}
```

- It's much simpler to "use" a namespace, in small programs there won't be any conflicts (and small is fairly big)

Compiling and linking, differences



It's all relative and it depends



Quadratic Equation Example

```
void Roots(double a, double b, double c,
     double& root1, double& root2);
// post: root1 and root2 set to roots of
//       quadratic ax^2 + bx + c
//       values undefined if no roots exist

int main()
{
    double a,b,c,r1,r2;
    cout << "enter coefficients ";
    cin >> a >> b >> c;
    Roots(a,b,c,r1,r2);

    cout << "roots are " << r1 << " " << r2 <<
  endl;
    return 0;
}
```


Who supplies memory, where's copy?

```
void Roots(double a, double b, double c,
     double& root1, double& root2);
// post: root1 and root2 set to roots of
//       quadratic ax^2 + bx + c
//       values undefined if no roots exist
```

- For value parameter, the argument value is copied into memory that "belongs" to parameter
- For reference parameter, the argument is the memory, the parameter *is an alias for argument memory*

```
double x, y, w, z;
Roots(1.0, 5.0, 6.0, x, y);
Roots(1.0, w, z, 2.0, x);    // no good, why?
```

Parameter Passing: const-reference

- When parameters pass information into a function, but the object passed doesn't change, it's ok to pass a copy
  - Pass by value means pass a copy
  - Memory belongs to parameter, argument is copied
- When parameter is altered, information goes out from the function via a parameter, a reference parameter is used
  - No copy is made when passing by reference
  - Memory belongs to argument, parameter is alias
- Sometimes we want to avoid the overhead of making the copy, but we don't want to allow the argument to be changed (by a malicious function, for example)
  - *const-reference* parameters avoid copies, but cannot be changed in the function

Count # occurrences of "e"

- Look at every character in the string, avoid copying the string

```
int letterCount(const string& s, const string& letter)
// post: return number of occurrences of letter in s
{
    int k, count = 0, len = s.length();
    for(k=0; k < len; k++) {
        if (s.substr(k,1) == letter) {
            count++;
        }
    }
    return count;
}
```

- Calls below are legal (but won't be if just reference parameters)

```
int ec = letterCount("elephant", "e");
string s = "hello"; cout << letterCount(s, "a");
```


General rules for Parameters

- Don't worry too much about efficiency at this stage of learning to program
  - You don't really know where efficiency bottlenecks are
  - You have time to develop expertise
- However, start good habits early in C++ programming
  - Built-in types: int, double, bool, char, pass by value unless returning/changing in a function
  - All other types, pass by const-reference unless returning/changing in a function
  - When returning/changing, use reference parameters
- Const-reference parameters allow constants to be passed, "hello" cannot be passed with reference, but ok const-reference

Rock Stars for Computer Science

STL concepts

- Container: stores objects, supports iteration over the objects
  - Containers may be accessible in different orders
  - Containers may support adding/removing elements
  - e.g., vector, map, set, deque, list, multiset, multimap
- Iterator: interface between container and algorithm
  - Point to objects and move through a range of objects
  - Many kinds: input, forward, random access, bidirectional
  - Syntax is pointer like, analagous to (low-level) arrays
- Algorithms
  - find, count, copy, sort, shuffle, reverse, …

Iterator specifics

- An iterator is dereferenceable, like a pointer
  - `*it` is the object an iterator points to
- An iterator accesses half-open ranges, [first..last), it can have a value of last, but then not dereferenceable
  - Analagous to built-in arrays as we'll see, one past end is ok
- An iterator can be incremented to move through its range
  - Past-the-end iterators not incrementable

```
vector<int> v; for(int k=0; k < 23; k++) v.push_back(k);
vector<int>::iterator it = v.begin();
while (it != v.end()) { cout << *v << endl; v++;}
```


STL overview

- STL implements generic programming in C++
  - Container classes, e.g., vector, stack, deque, set, map
  - Algorithms, e.g., search, sort, find, unique, match, …
  - Iterators: pointers to beginning and one past the end
  - Function objects: less, greater, comparators
- Algorithms and containers decoupled, connected by iterators
  - Why is decoupling good?
  - Extensible: create new algorithms, new containers, new iterators, etc.
  - Syntax of iterators reflects array/pointer origins, an array can be used as an iterator

STL examples: wordlines.cpp

- How does an iterator work?
  - Start at beginning, iterate until end: use [first..last) interval
  - Pointer syntax to access element and make progress

```
vector<int> v; // push elements
vector<int>::iterator first = v.begin();
vector<int>::iterator last  = v.end();
while (first < last) {
    cout << *first << endl;
    ++first;
}
```

  - Will the while loop work with an array/pointer?
- In practice, iterators aren't always explicitly defined, but passed as arguments to other STL functions

Review: what's a map, a set, a …

- Maps keys to values
  - Insert key/value pair
  - Extract value given a key, iterate over pairs
  - STL uses red-black tree, guaranteed O(log n) …
    - STL unofficially has a hash_map, see SGI website
  - Performance and other trade-offs?
- A set can be implemented by a map
  - Stores no duplicates, in STL guaranteed O(log n), why?
  - STL also has multimap

arrays and strings: what's a char *?

- Why not rely solely on string and vector classes?
  - how are string and vector implemented?
  - lower level access can be more efficient (but be leery of claims that C-style arrays/strings *required* for efficiency)
  - real understood comes when more levels of abstraction are understood
- string and vector classes insulate programmers from inadvertent attempts to access memory that's not accessible
  - what is the value of a pointer?
  - what is a segmentation violation?

Contiguous chunks of memory

- In C++ allocate using array form of new

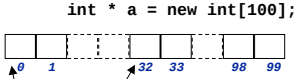
```
int * a = new int[100];
double * b = new double[300];
```

- new [] returns a pointer to a block of memory
  - how big? where?
- size of chunk can be set at runtime, not the case with

```
int a[100];
cin >> howBig;
int a[howBig];
```

- delete [] a; // storage returned

```
int * a = new int[100];
```




```
a is a pointer
*a is an int
a[0] is an int (same as *a)
a[1] is an int
a+1 is a pointer
a+32 is a pointer
*(a+1) is an int (same as a[1])
*(a+99) is an int
*(a+100) is trouble
a+100 is valid for comparison
      of pointer values
```

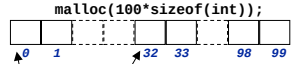

C-style contiguous chunks of memory

- In C, malloc is used to allocate memory

```
int * a = (int *)
   malloc(100 * sizeof(int));
double * d = (double *)
   malloc(200 * sizeof(double));
```

- malloc must be cast, is NOT type-safe (returns void *)
  - void * is 'generic' type, can be cast to any pointer type
- free(d); // return storage
- We WILL NOT USE malloc/free

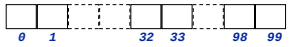
```
int * a = (int *)
      malloc(100*sizeof(int));
```




```
a is a pointer
*a is an int
a[0] is an int (same as *a)
a[1] is an int
a+1 is a pointer
a+32 is a pointer
*(a+1) is an int (same as a[1])
*(a+99) is an int
*(a+100) is trouble
a+100 is valid for comparison
```

Address calculations, what is sizeof(…)?

```
int * a = new int[100];
```




```
a[33] is the same as *(a+33)
if a is 0x00a0, then a+1 is
0x00a4, a+2 is 0x00a8
(think 160, 164, 168)
```

```
double * d = new double[200];
```


```
*(d+33) is the same as d[33]
if d is 0x00b0, then d+1 is
0x00b8, d+2 is 0x00c0
(think 176, 184, 192)
```

- x is a pointer, what is x+33?
  - a pointer, but where?
  - what does calculation depend on?
- result of adding an int to a pointer depends on size of object pointed to
- result of subtracting two pointers is an int:

```
(d + 3) - d == _______
```


Who is Alan Perlis?

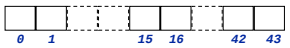
- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not precede complexity, but follows it
- If you have a procedure with ten parameters you probably missed some
- If a listener nods his head when you're explaining your program, wake him up
- Programming is an unnatural act
- Won first Turing award

http://www.cs.yale.edu/homes/perlis-alan/quotes.html

More pointer arithmetic

- address one past the end of an array is ok for *pointer comparison only*
- what about *(begin+44)?
- what does begin++ mean?
- how are pointers compared using < and using == ?
- what is value of end - begin?

```
char * a = new char[44];
char * begin = a;
char * end = a + 44;
```




```
while (begin < end)
{
   *begin = 'z';
   begin++;  // *begin++ = 'z'
}
```


What is a C-style string?

- array of char terminated by sentinel '\0' char
  - sentinel char facilitates string functions
  - '\0' is nul char, unfortunate terminology
  - how big an array is needed for string "hello"?
- a string is a pointer to the first character just as an array is a pointer to the first element
  - `char * s = new char[6];`
  - what is the value of s? of s[0]?
- char * string functions in <string.h>

C style strings/string functions

- strlen is the # of characters in a string
  - same as # elements in char array?

```
int strlen(char * s)
// pre: '\0' terminated
// post: returns # chars
{
    int count=0;
    while (*s++) count++;
    return count;
}
```

- Are these less cryptic?

```
while (s[count]) count++;
// OR, is this right?
char * t = s;
while (*t++);
return t-s;
```

- what's "wrong" with this code?

```
int countQs(char * s)
// pre: '\0' terminated
// post: returns # q's
{
  int count=0;
  for(k=0;k <
  strlen(s);k++)
     if (s[k]=='q')
  count++;
  return count;
}
```

- how many chars examined for 10 character string?
- solution?

<string.h> aka <cstring> functions

- strcpy copies strings
  - who supplies storage?
  - what's wrong with s = t?

```
char s[5];
char t[6];
char * h = "hello";
strcpy(s,h); // trouble!
strcpy(t,h); // ok

char * strcpy(char* t,char* s)
//pre: t, target, has space
//post: copies s to t,returns t
{
   int k=0;
   while (t[k] = s[k]) k++;
   return t;
}
```

- strncpy copies n chars (safer?)
- what about relational operators <, ==, etc.?
- can't overload operators for pointers, no overloaded operators in C
- strcmp (also strncmp)
  - return 0 if equal
  - return neg if lhs < rhs
  - return pos if lhs > rhs

```
if (strcmp(s,t)==0) // equal
if (strcmp(s,t) < 0)// less
if (strcmp(s,t) > 0)// ????
```


Arrays and pointers

- These definitions are related, but not the same

```
int a[100];
int * ap = new int[10];
```

- both a and ap represent 'arrays', but ap is an lvalue
- arrays converted to pointers for function calls:

```
char s[] = "hello";
// prototype: int strlen(char * sp);
cout << strlen(s) << endl;
```

- multidimensional arrays and arrays of arrays

```
int a[20][5];
int * b[10]; for(k=0; k < 10; k++) b[k] = new int[30];
```