

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-existent (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 language**
 - 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`

C++ idioms/general concepts

- **Genericity**
 - Templates, STL, containers, algorithms
- **Copy/Assignment/Memory**
 - Deep copy model, memory management “required”
- **Low-level structures**
 - C-style arrays and strings compared to STL, Tapestry
- **const**
 - Good for clients, bad for designers/coders?
- **From C to C++ to Java**
 - function pointers, function objects, inheritance

Language Design Goals

- **Java**
 - **New, pure O-O language**
 - **Compiled to byte-code, write once run “anywhere”**
simple, object-oriented, portable, interpreted, robust, secure, architecture-neutral, distributed, dynamic, multithreaded, high performance
- **C++**
 - **Compatible, O-O language features added on to C**
 - **No feature was included that would**
 - *cause a serious incompatibility with C*
 - *cause run-time or space overheads for a program that didn't use it*
 - *increase run-time or space requirements for a C program.*
 - *significantly increase the compile time compared*
 - *not be easy and efficient to implement in a traditional C programming environment.*

How do we read a file? (SearchDemo)

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

```
C++:   set<string> unique;
int total = 0;
string word;
while (input >> word){
    transform(word.begin(), word.end(),
              word.begin(), ::tolower);
    unique.insert(word);
    total++;
}
myWords = vector<string>(unique.begin(), unique.end());
```

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.

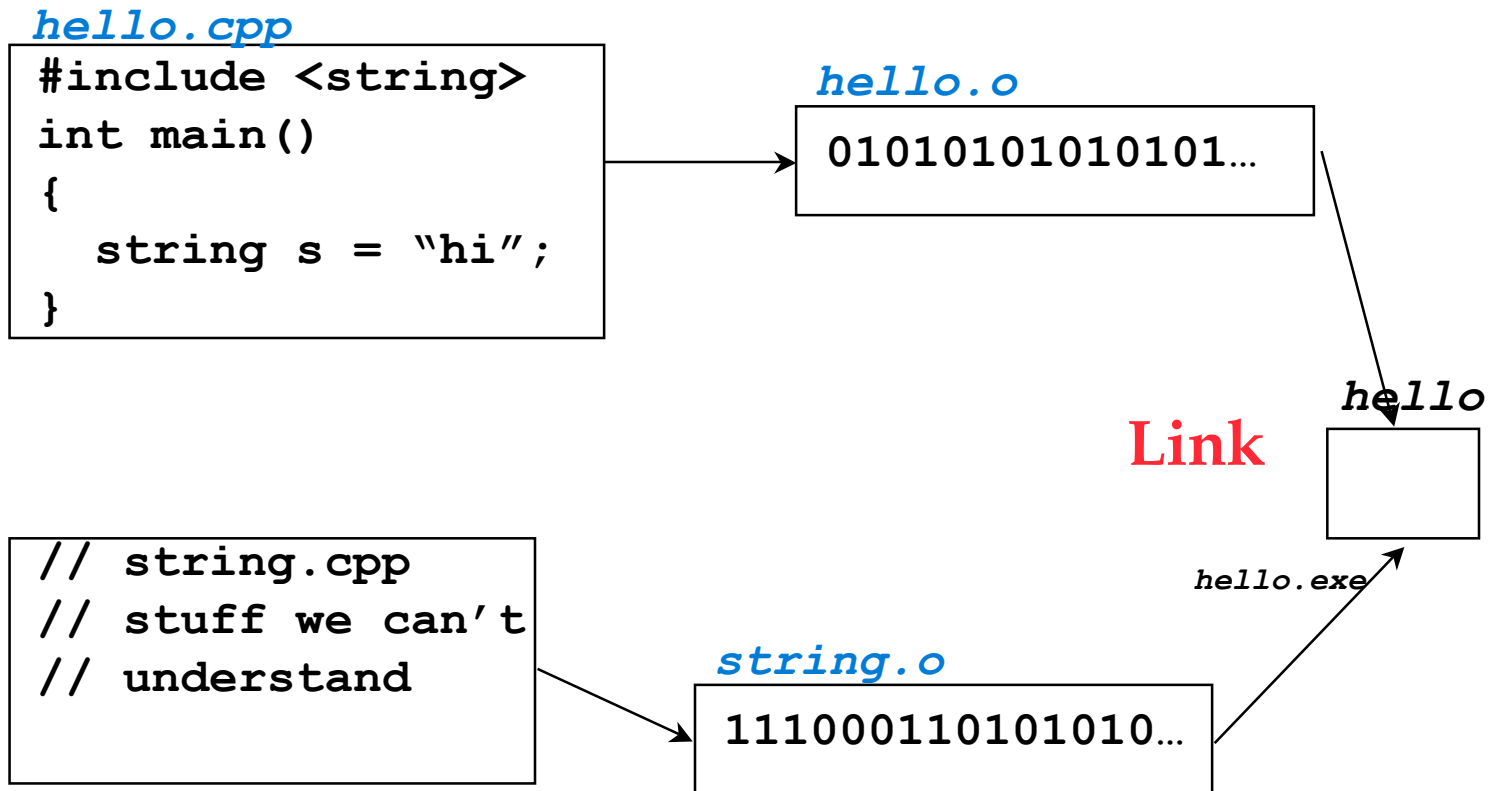
```
#include <iostream>

// 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



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 count = 0;
    for(int k = 0; k < s.length(); 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