Interlude – C++

Lecture 2

Hartmut Kaiser

https://teaching.hkaiser.org/spring2025/csc4700/

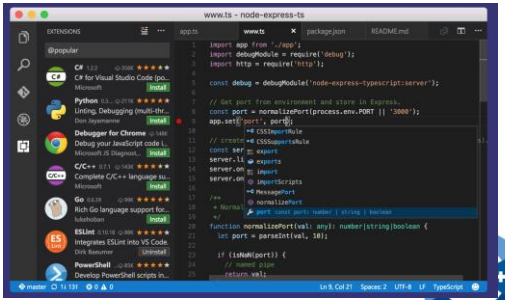
Write A Program in C++

- High performance

- Measure and Tune program
- Run program
- Compile program
- Write program

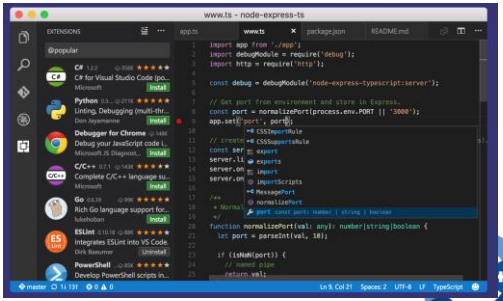
Developing your Code

- A computer should be a labor saving device
  - That includes (especially) mental labor
- Use productivity tools
- VS code (recommended)
  - Many others exist: Atom, Eclipse …



The CSC470 Development Environment

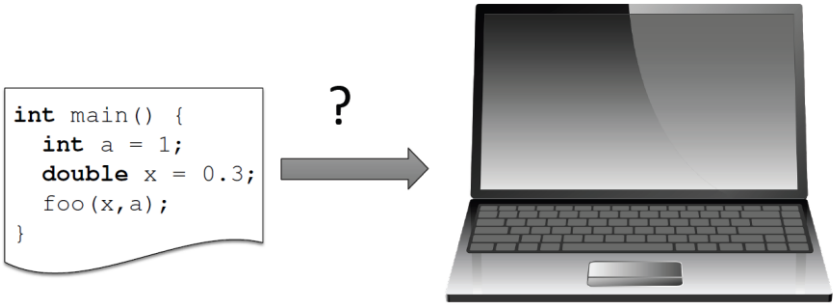
- Windows and Mac OS X can be made to approximate Linux
  - Uniform docker environment for everyone
  - Alternatively, use Github codespaces
- How-to documentation in assignment instructions online



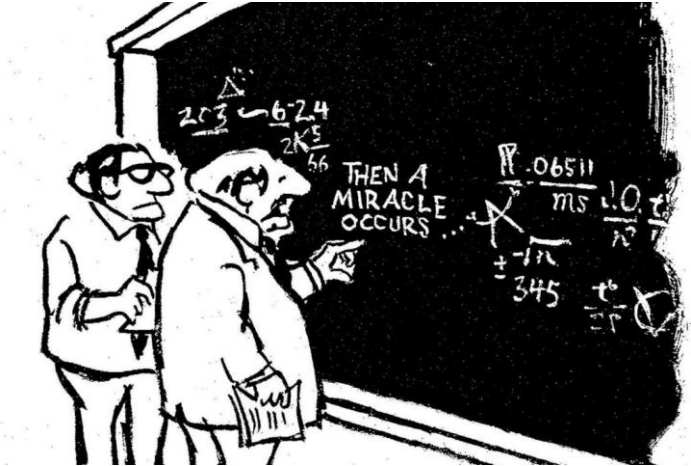
Linux shells

- sh: “Bourne shell” (Stephen Bourne, Bell Labs c.1977)
- ksh: Korn shell (David Korn, Bell Labs, c. 1983)
- csh: C shell (Bill Joy, UC Berkeley, 70s)
  - and cousin tcsh
- bash (Brian Fox, 1989)
  - Who knows what this stands for (without searching)?
- All are Linux (Unix) processes with read-eval-print loops
- But also complete systems scripting language for dealing with Unix
  - Unix philosophy: data in text format, small programs using text I/O

Programs and Programming



Programs and Programming

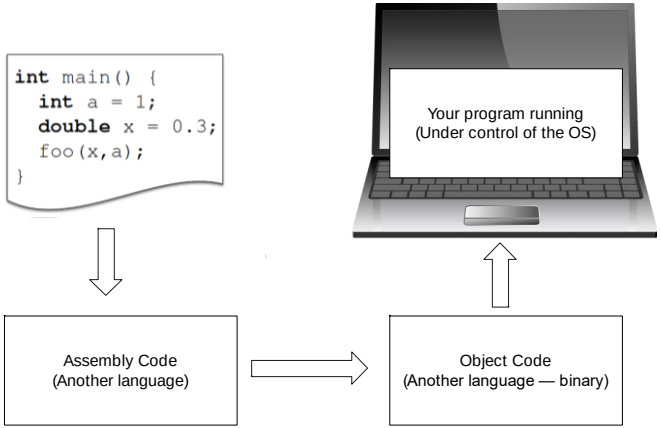


© S. Harris

Interpreted Language (Python)

```
import math

a = 3.14
b = math.sqrt(a)
print(b)
```

Compiled Language (C++)



Interpreted vs. Compiled Language

```python
import math

a = 3.14
b = math.sqrt(a)
print(b)
```

```cpp
#include <cmath>
#include <print>

int main()
{
    double a = 3.14;
    double b = std::sqrt(a);

    print("{}", b);

    return 0;
}
```

Compilation

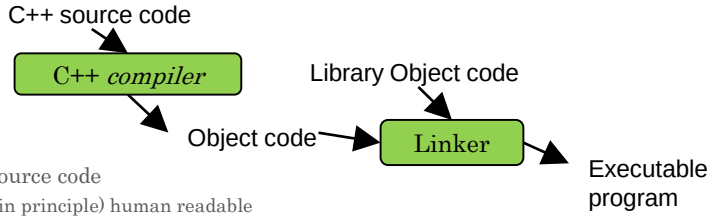
- You can't run (directly execute) this code
- It needs to be turned into code that can run
  - An executable
- Multi-step process
  - Compile to object file
    - Bits just for this code
  - Then link in libraries for sqrt and IO

```
#include <cmath>
#include <print>

int main()
{
    double a = 3.14;
    double b = std::sqrt(a);

    std::print("{}", b);

    return 0;
}
```

Compiling

- To compile one source file to an executable
  - $ c++ filename.cpp
  - (What is the name of the executable?)
- To compile multiple source files to an executable
  - $ c++ one.cpp two.cpp three.cpp
- To create an object file
  - $ c++ -c one.cpp –o one.o
- To create an executable from multiple object files
  - $ c++ one.o two.o three.o –o myexecutable

Compilation and Linking



- You write C++ source code
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  - Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  - E.g. input/output libraries, operating system code, and windowing code
- The result is an executable program
  - E.g. a .exe file on windows or an a.out file on Unix

See: Decoding C++ Compilation Process: From Source Code to Binary

CMake

- CMake is a family of tools
  - Building software
  - Testing software
  - Packaging software
- We will use it for building and testing
- CMake generates build system files (Makefiles and or workspaces)
  - Those can be used to automatically build and test your code
- The user writes a single set of descriptive scripts
  - Define Targets and their inter-dependencies
- CMake is well integrated with many IDEs (also VSCode)

Simplest Example

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.0)
project(Demo1)
add_executable(Demo1 demo1.cpp)
```

In source build:

```
% cd demo1
% ls
CMakeLists.txt  demo1.cpp
```

Simplest Example

```
% cmake .
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile fe
-- Detecting C compile fe
-- Check for working CXX
-- Check for working CXX
-- Detecting CXX compiler
-- Detecting CXX compiler
-- Detecting CXX compile
-- Detecting CXX compile
-- Configuring done
-- Generating done
-- Build files have been
```

```
% make
Scanning dependencies of target Demo1
[ 50%] Building CXX object
CMakeFiles/Demo1.dir/demo1.cpp.o
[100%] Linking CXX executable Demo1
[100%] Built target Demo1

% ./Demo1
Hello, world!
```

Slice of C++

- C++11 (C++14, C++17, C++20, C++23) are quite modern languages
- But C++11 (et. Al.) and libraries are huge
- We will use a focused slice of C++23
- Use some modern features
- Avoid legacy features (such as pointers)
- Avoid modern features (OO)

```
#include <cmath>
#include <print>

int main()
{
    double a = 3.14;
    double b = std::sqrt(a);

    std::print("{}", b);

    return 0;
}
```

Hello World!

- `#include` statement
- Function definition, here:
  - Takes no arguments
  - Returns an `int` (integer)
- Code block – scoping!
  - `std` namespace
  - Character string
  - Returns `0` (zero, for main it means success)
- Comments use '`//`' or '`/*...*/`'

```
#include <print>

// This is a minimal example
int main()
{
    std::print("Hello World!");
    return 0;
}
```

Types

- Variable definition

```
std::string contents;
int x;
double y;
```

- C++ has many built-in types: `int`, `double`, `char`, etc.
- Other types are defined for libraries (accessed via `#include`)
- Almost always class definitions

Declaring and Initializing Variables

- In the old days variables were declared at the beginning of a block

```
int main() {
  double x, y;
  // ...
  x = 3.14159;
  y = x * 2.0;
  // ...
  return 0;
}
```

Now they can be defined anywhere in the block

```
int main() {
  // ...
  double x = 3.14159;
  double y = x * 2.0;
  // ...
  return 0;
}
```

- Best practice: Don't declare variables before they are needed and **always** initialize if possible

Namespaces

- Provides a scope to the identifiers (the names of types, functions, variables, etc.) inside it
- Organize code into logical groups
- Prevent name collisions that can occur especially when your code base includes multiple libraries

Namespaces

```
#include <cmath>
#include <print>

double pi = 3.14;

int main()
{
    std::print("The value of pi is: {}", pi);
    return 0;
}
```

- To print the value `pi`
  - The compiler needs to resolve (find) the symbol (variable)

Namespaces

```
#include <cmath>
#include <print>

namespace csc4700 {
    double pi = 3.14;
}

int main()
{
    std::print("The value of pi is: {}", csc4700::pi);
    return 0;
}
```

- To print the value `pi`
  - The compiler needs to resolve (find) the symbol (variable)
- Here `pi` is defined in a namespace `csc4700`
  - We need to tell the compiler where to look

Namespaces

```
#include <cmath>
#include <print>

namespace csc4700 {
    double pi = 3.14;
}

int main()
{
    using namespace csc4700;
    std::print("The value of pi is: {}", pi);
    return 0;
}
```

- To print the value `pi`
  - The compiler needs to resolve (find) the symbol (variable)
- Here `pi` is defined in a namespace `csc4700`
  - We need to tell the compiler where to look
  - We can also hoist everything from one namespace into the current

Namespaces

```
namespace csc4700 {
    double pi = 3.14;
}
namespace csc4700_1 {
    double pi = 3.14;
}

int main()
{
    using namespace csc4700;
    std::print("The value of pi is: {}", pi);
    return 0;
}
```

- To print the value `pi`
  - The compiler needs to resolve (find) the symbol (variable)
- Here `pi` is defined in a namespace `csc4700` and `csc4700_1`
  - Requires disambiguation

Organizing your Programs

- Software development is difficult
- How do humans attack complex problems?
  - Abstract details
  - Modular / reusable
  - Well defined interfaces and functionality
  - Understandable
- Apply the same principles to software

Newton's Method for Square Root

- To solve $f(x) = 0$ for x
- Linearize (approximate the nonlinear problem with a linear one) and solve the linear problem
- Iterate: $f(x + \Delta x) \approx f(x) + \Delta x f'(x)$
- Taylor: $\Delta x = -\frac{f(x)}{f'(x)}$
- Example:

$$f(x) = x^2 - y = 0 \rightarrow y = \sqrt{x}$$
$$f'(x) = 2x \rightarrow \Delta x = -\frac{x^2 - y}{2x}$$

Compute Square Root of 2

```cpp
#include <cmath>
#include <print>

int main()
{
    double x = 1.0;
    for (size_t i = 0; i != 32; ++i)
    {
        double dx = - (x*x - 2.0) / 2.0 * x;
        x += dx;
        if (std::abs(dx) <= 1.e-9)
            break;
    }
    std::print("The square root of 2 is: {}", x);
}
```

Compute Square Root of 3

```
#include <cmath>
#include <print>

int main()
{
    double x = 1.0;
    for (size_t i = 0; i != 32; ++i)
    {
        double dx = - (x*x - 3.0) / 2.0 * x;
        x += dx;
        if (std::abs(dx) <= 1.e-9)
            break;
    }
    std::print("The square root of 2 is: {}", x);
}
```

Compute Square Root of Anything

```
namespace csc4700 {
    double sqrt(double y) {
        double x = 1.0;
        for (size_t i = 0; i != 32; ++i) {
            double dx = - (x*x - y) / 2.0 * x;
            x += dx;
            if (std::abs(dx) <= 1.e-9)
                break;
        }
        return x;
    }
}

int main() {
    std::print("The square root of 2 is: {}",
        csc4700::sqrt(2.0));
}
```

Experiment: what will it print?

```
namespace csc4700 {
    double sqrt(double y) {
        double x = 1.0;
        for (size_t i = 0; i != 32; ++i) {
            double dx = - (sqr(x) - 2.0) / 2.0 * x;
            x += dx;
            if (std::abs(dx) <= 1.e-9) break;
        }
        y = x;        // change value of argument
        return x;
    }
}

int main() {
    double y = 2.0;
    std::print("sqrt(2) is: {}", csc4700::sqrt(y));
    std::print("y is: {}", y);
}
```

Parameter Passing in C++

- The argument y was passed by value
  - The function will operate on a copy of the original argument
  - Only the copy is changed, not the original
- C++ has “pass by value” semantics
- Just to be clear, the parameter can have any name (don’t confuse with y declared in main)

Experiment: what will it print?

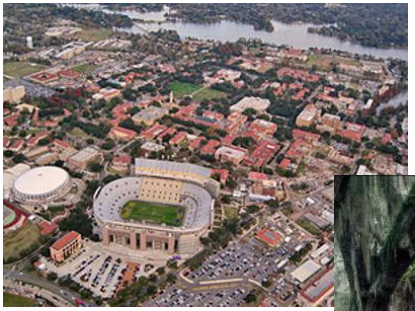
```
namespace csc4700 {
    double sqrt(double& y) {   // <-- added '&'
        double x = 1.0;
        for (size_t i = 0; i != 32; ++i) {
            double dx = - (sqr(x) - 2.0) / 2.0 * x;
            x += dx;
            if (std::abs(dx) <= 1.e-9) break;
        }
        y = x;        // change value of argument
        return x;
    }
}

int main() {
    double y = 2.0;
    std::print("sqrt(2) is: {}", csc4700::sqrt(y));
    std::print("y is: {}", y);
}
```

Parameter Passing in C++

- The argument `y` was now passed by reference
  - The function will operate on the original argument
- Both variables refer to the same instance of the double!
- Why would we want to pass a reference?
  - "Out parameters"
  - Efficiency (no copy)
- Pass by `const` reference is used to promise not to change the original argument:
  - `double sqrt(double const& y) {}`
  - Still gives the benefit of efficiency, if needed

C++ Core Guidelines: Functions

- F.2: A function should perform a single logical operation
- F.3: Keep functions short and simple
- F.16: For “in” parameters, pass cheaply-copied types by value and others by reference to const
- F.17: For “in-out” parameters, pass by reference to non-const
- F.20: For “out” output values, prefer return values to output parameters



LSU CENTER FOR COMPUTATION & TECHNOLOGY

STE||AR GROUP