CS-202
Course Introduction
C++ Primer – GNU Compilation starters

C. Papachristos

Autonomous Robots Lab University of Nevada, Reno



Your Instructor

Christos Papachristos

- Education
 MS in Electrical & Computer Engineering
 PhD in Autonomous (Aerial) Robotics
 Post-Doc Research UNR
- ➤ Active Areas

 Robotics Research @ Autonomous Robots Lab (ARL) UNR

 Coding / SW Engineering for Field Robotics
- Find me at:

 ARF118 / ARF004 / Autonomous Robots ARENA (a.k.a "the HighBay")

 cpapachristos@unr.edu

Course Requirements

> CS-135 (@least C- grade)

Basic program design

Control structures for s

Control structures for selection, iteration

Use functions, employ/implement prototypes, pass parameters

Multi-dimensional arrays (declaration/usage/passing as parameters)

Basic stream I/O (with external file usage)

Course Objectives

Introductory-level experience to:

- Object-oriented programming (OOP)
- Object-oriented design (**OOD**)

Basic software engineering techniques, Proper program design principles

Pointers, Dynamic Memory

C++ libraries, Classes – operators, the STL

(Coding) Tools:

- C++ programming language
- GCC (GNU Compiler)
- The GNU Make tool
- Linux Operating System

Course Rules

Grading Policy (tentative):

Component	Percentage	
Projects & Assignments	50% (Breakdown: 40% – 10% each)	
Midterm Exam	20%	
Final Exam	30%	

You cannot earn a passing grade in the course without a passing grade on both:

- The average of the Midterm and Final exams.
- The average of the Projects.

Grade re-scaling (plus/minus) may be assigned based on an outstanding or inferior Final exam.

Class presence will be required, tracked, and factored in, for the course's Lab Sections. For general university policy regarding class absence, see <u>UAM 3,020</u>.

Course Rules

Tasks & Responsibilities:

Weekly Projects – Bi-weekly MAX

Turn in via WebCampus! LATE submission (24hrs max) incurs 20% penalty.

After 24hrs NO SUBMISSION WILL BE ACCEPTED.

Lab Assignments – In-Lab! NO SUBMISSION WILL BE ACCEPTED AFTER.

Academic Dishonesty:

Cheating, plagiarism or otherwise obtaining grades under false pretenses constitute academic dishonesty according to the code of this university. Academic dishonesty will not be tolerated and penalties can include filing a final grade of "F"; reducing the student's final course grade one or two full grade points; awarding a failing mark on the coursework in question; or requiring the student to retake or resubmit the coursework. For more details, see the <u>University of Nevada, Reno General Catalog</u>. (Also, refer to Academic Standards in course syllabus and <u>online</u>)

Course Help

Course & Projects:

Peer-Assisted Study Session (PASS) Leader:

Kurtis Rodrigue kurtisr@nevada.unr.edu

3 hrs in-class, 4 hrs PASS sessions

Teaching Assistants:

Bahadir Ali, Spencer Gibb, Jun Yi <u>bpehlivan@</u>, <u>sgibb@</u>, <u>junyi@</u> [nevada.unr.edu] 50 mins per Lab section, Tutoring & quizzes

Disability Services:

Any student with a disability needing academic adjustments or accommodations is requested to speak with the Disability Resource Center as soon as possible to arrange for appropriate accommodations.

Course Week

Course, Projects, Labs:

Monday	Tuesday	Wednesday	Thursday	Friday
			Lab (9:00-12:50)	
	CLASS		CLASS	
PASS	PASS	Project DEADLINE	NEW Project	
Session	Session		INL VV PTOJECT	

Projects will start this Thursday 1/31.

Labs will start **this** Thursday 1/31.

Today's Concepts

Differences between programming languages

C++ vs Python

Compiled vs Interpreted

Programming "style" restrictions

C++ concepts (Learning C++ is only part of the deal)

Classes

Object-Oriented Programing

Object-Oriented Design

(But Remember: C++ is the "Latin" of programming languages)

Today's Practices

Using the Linux Operating System.

Developing under the GCC (GNU Compiler Collection) suite.

- Linux basics, Shell usage introduction.
- Text editor for code writing emacs / gedit / nano / vim.

(IDEs such as Eclipse, XCode, Codeblocks etc. are not part of the supported toolchains, your programs will have to compile & run on installations like the ones on the machines of SEM321).

- Compilation toolchain invocation.
- Basics for large (sort of) software development project setup.

Code – Compile – Run

Challenges

Getting started:

The Linux environment (some transfer students could have a hard time).

Start with your Projects early on.

Don't play down the significance of the Lab sessions.

Don't wait to seek help. Benefit from all sources.

Your TAs.

Your PASS Leader.

WebCampus discussions.

The web... http://en.cppreference.com/w/cpp

http://www.cplusplus.com/reference/

C++ Basic Info

Created in 1979 by Bjarne Stroustrup – *the originator*. At Bell Labs (home of UNIX and C).

The story: "Invent a computer programming language so arcane and complex that no one except him would be able to use it."

The background: "Languages like Simula and Ada."

The joke: "Colleagues laughed a his first cut at an inscrutably hard language. Then, he went back to his lair and didn't emerge until he had added references and templates."

C++ Basic Info

Added object-oriented features to C.

Renamed to C++ in honor of auto-increment operator.

Later standardized with several International Organization for Standards (ISO) specifications.

Constantly expanding/updating standards: <

Greatly influenced Java development (1991).

- Popular modern OO language
- ➤ Wide industry usage
- Used in many types of applications
- > Object-Oriented
- Portable (not as much as Java, but fairly so)
- > Efficient
- Retains much of its C origins

C++98 C++03 C++11 C++14

Procedural vs Object-Oriented

Procedural

Modular units: functions

Program structure: hierarchical

Data and operations are not bound

to each other

Examples:

C, Pascal, Basic, Python

Object-Oriented (OO)

Modular units: objects

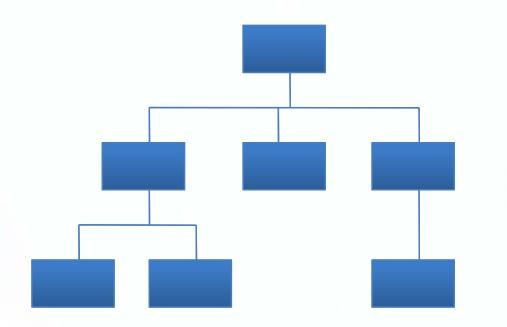
Program structure: a graph

Data and operations are bound to each

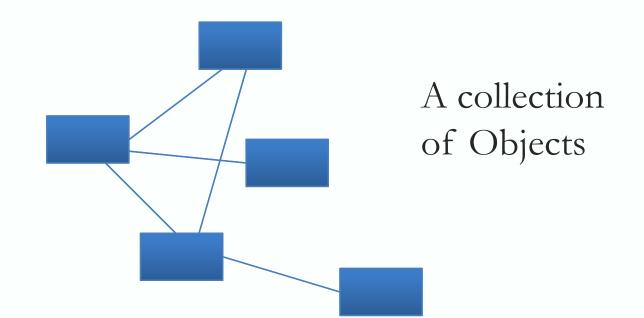
other

Examples:

C++, Java, Python

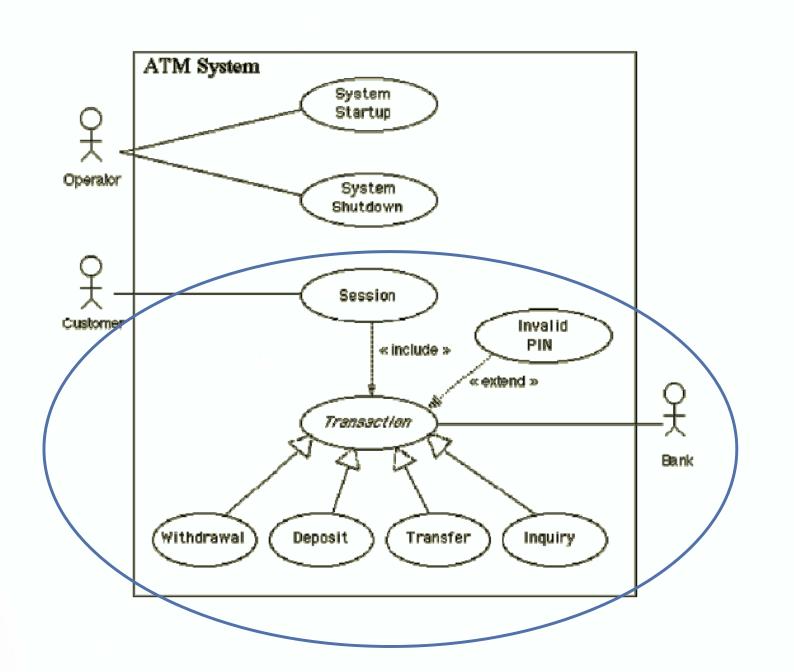


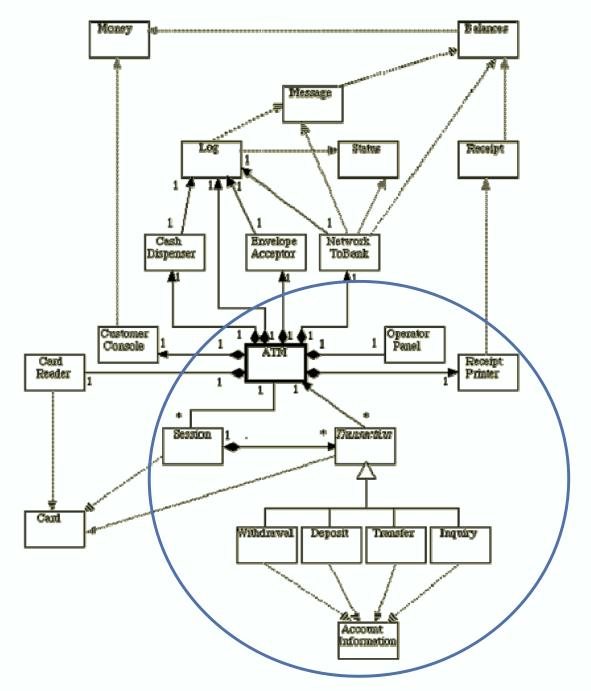
A hierarchy of functions



Procedural vs Object-Oriented

The ATM Machine paradigm



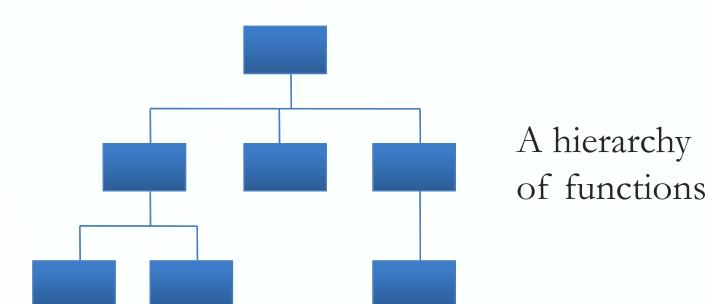


Procedural vs Object-Oriented

Procedural

Focused on the question: "What should the program do next?" Structure program by:

- > Splitting into sets of tasks and subtasks.
- Make functions for tasks.
- Perform them in sequence (computer). Large amount of data and/or tasks makes projects/programs unmaintainable.



Object-Oriented (OO)

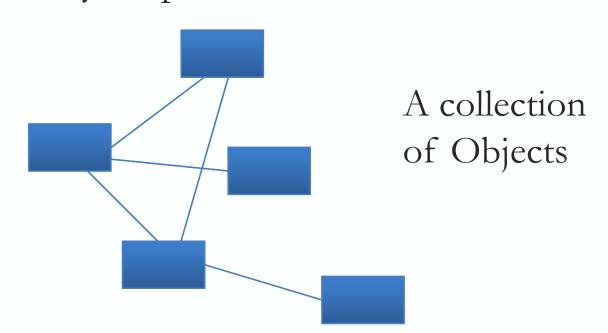
Package-up self-sufficient modular pieces of code.

The world is made up of interacting objects.

Pack away details into boxes (objects) keep them in mind in their abstract form.

Focus on (numerous) interactions.

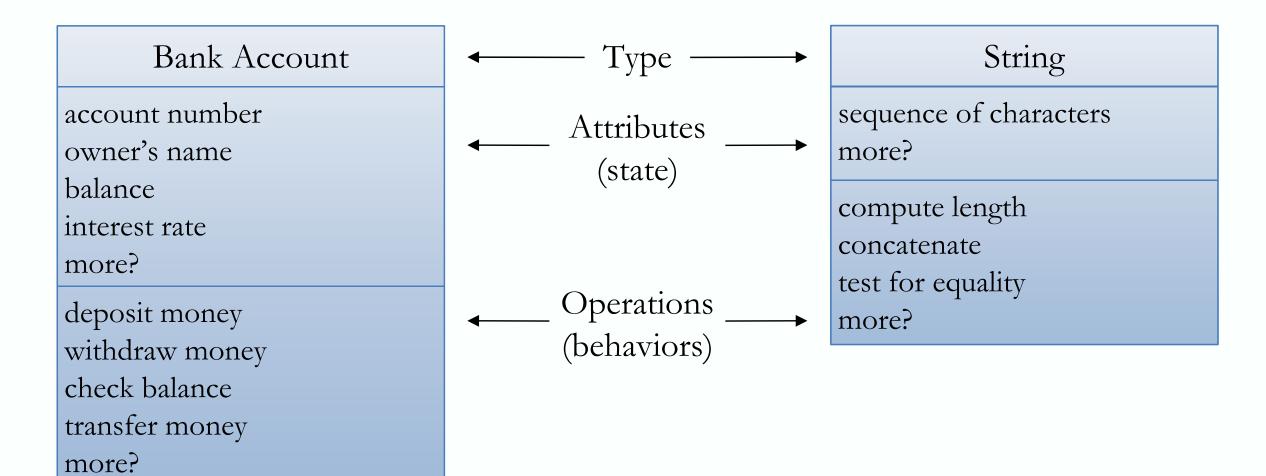
- Encapsulation
- > Inheritance
- Polymorphism



Classes

Class

C++ Classes are very similar to C Structs, in that they both include user-defined sets of data items, which collectively describe some entity such as a Student, Book, or Airplane, etc...



Classes

Class

A data type containing:

more?

- ➤ Attributes make up the object's state
- > Operations define the object's behaviors

Bank Account	← Type →	String
account number owner's name	← Attributes ← (state)	sequence of characters more?
balance interest rate more?		compute length concatenate test for equality
deposit money withdraw money	Operations (behaviors)	more?
check balance transfer money		

Object

A particular instance of a class.

My Account

12-345-6 Christos Papachristos \$1,250.86 1.5%

Chang's Account

65-432-1 Richard Chang \$5.50 2.7%

Gibson's Account

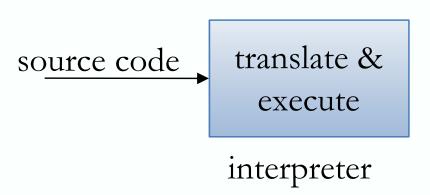
43-261-5 Katherine Gibson \$825.50 2.5%

For any of these accounts, one can...

- Deposit money
- Withdraw money
- Check the balance
- Transfer money

Interpreters, Compilers, Hybrids

Interpreted Languages (e.g. JavaScript, Perl, Ruby)

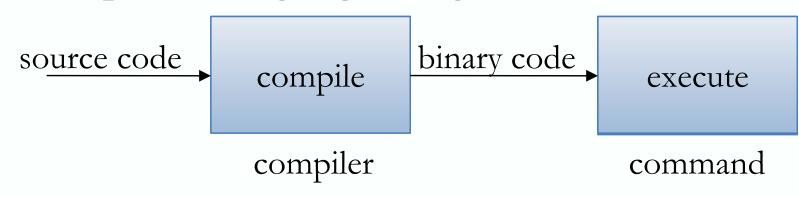


Interpreter translates source into binary and executes it.

Small, easy to write

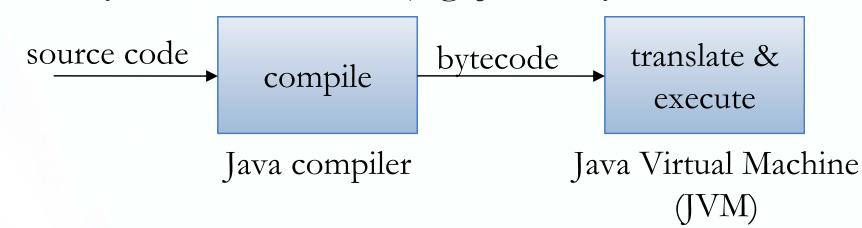
Interpreter is a *program* unique to each platform (operating system)

Compiled Languages (e.g. C, C++)



Compiler is platform dependent.
Code once compiled is expressed in the *instructions* of the target machine.

Many other models: (e.g. Java, Python even stranger)



Bytecode is platform independent
Java *Virtual Machine* (the target platform) is an interpreter that is platform dependent

CS-202 C. Papachristos



```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

These pieces of code do the same thing! What's different about these two languages?

```
C++
#include <iostream>
using namespace std;
int main() {
   int quotient ;
   cout << "Hello, world" << endl ;</pre>
   quotient = 3 / 4;
   if (quotient == 0) {
     cout << "3/4 == 0";
     cout << " in C++" ;
   } else {
     cout << "3/4 != 0";
   return 0;
```

```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

Elements of C++

Must have a "main()" function Statements end with ";" Variables must be declared "if/else" syntax different Statement blocks demarcated by "{...}"

```
C++
#include <iostream>
using namespace std;
int main() {
   int quotient ;
   cout << "Hello, world" << endl ;</pre>
   quotient = 3 / 4;
   if (quotient == 0) {
     cout << "3/4 == 0";
     cout << " in C++" ;
   } else {
     cout << "3/4 != 0";
   return 0;
```

```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

Elements of C++

Must have a "main()" function Statements end with ";" Variables must be declared "if/else" syntax different Statement blocks demarcated by "{...}"

```
C++
#include <iostream>
using namespace std;
int main() {
   int quotient ;
   cout << "Hello, world" << endl ;</pre>
   quotient = 3 / 4 ;
   if (quotient == 0) {
     cout << "3/4 == 0" ;
     cout << " in C++" ;
   } else {
     cout << "3/4 != 0" ;
   return 0;
```

```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

Elements of C++

Must have a "main()" function Statements end with ";" Variables must be declared "if/else" syntax different Statement blocks demarcated by "{...}"

```
C++
#include <iostream>
 using namespace std;
 int main() {
   int quotient ;
   cout << "Hello, world" << endl ;</pre>
   quotient = 3 / 4;
   if (quotient == 0) {
     cout << "3/4 == 0";
     cout << " in C++" ;
   } else {|
     cout << "3/4 != 0";
   return 0 ;
```

```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
   print "3/4 == 0",
   print "in Python"
else:
   print "3/4 != 0"
```

These pieces of code do the same thing! What's different about these two languages?

```
C++
#include <iostream>
using namespace std;
int main() {
   int quotient ;
   cout << "Hello, world" << endl ;</pre>
   quotient = 3 / 4;
   if (quotient == 0) {
    cout << "3/4 == 0";
    |cout << " in C++" ;
   } else {
     cout << "3/4 != 0";
   return 0;
```

```
print "Hello, world"
quotient = 3.0 / 4
if quotient == 0:
   print "3.0/4 == 0",
   print "in Python"
else:
   print "3.0/4 != 0"
```

Extra Look?

3.0 / 4

What's different about these two languages?

```
C++
#include <iostream>
using namespace std;
int main() {
  int quotient ;
  cout << "Hello, world" << endl ;</pre>
  quotient = 3.0 / 4 ;
  if (quotient == 0) {
   cout << "3.0/4 == 0";
   |cout << " in C++" ;
  } else {
    cout << "3.0/4 != 0";
  return 0;
```

```
print "Hello, world"
quotient = 3.0 / 4
if quotient == 0:
   print "3/4 == 0",
   print "in Python"
else:
   print "3/4 != 0"
```

Extra Look?

```
float quotient ;
```

What's different about these two languages?

```
C++
#include <iostream>
using namespace std;
int main() {
 float quotient;
  cout << "Hello, world" << endl ;</pre>
  quotient = 3.0 / 4;
  if (quotient == 0) \{
    cout << "3.0/4 == 0";
    cout << " in C++" ;
  } else {
   cout << "3.0/4 != 0";
  return 0;
```

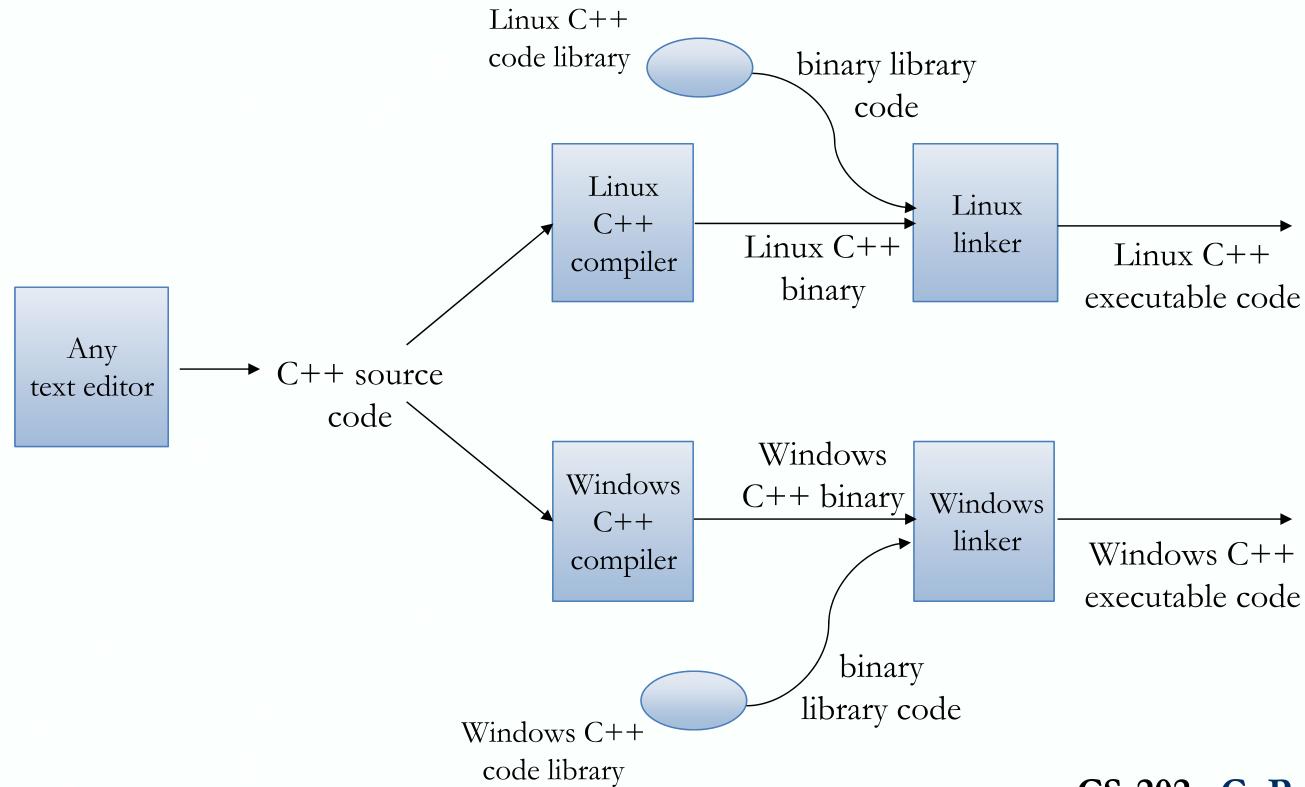
```
print "Hello, world"
quotient = 3 / 4
if quotient == 0:
   print "3/4 == 0",
   print "in Python"
else:
   print "3/4 != 0"
```

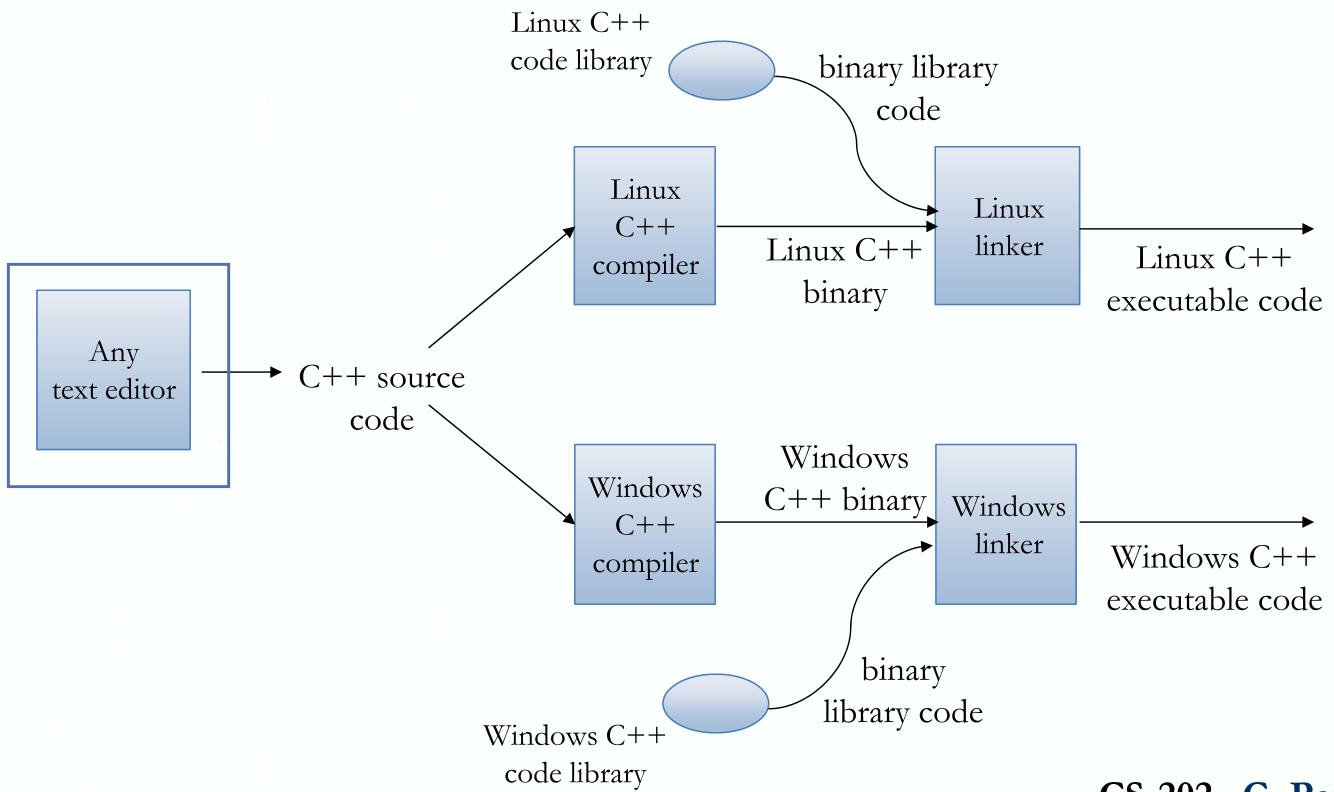
Extra Look?

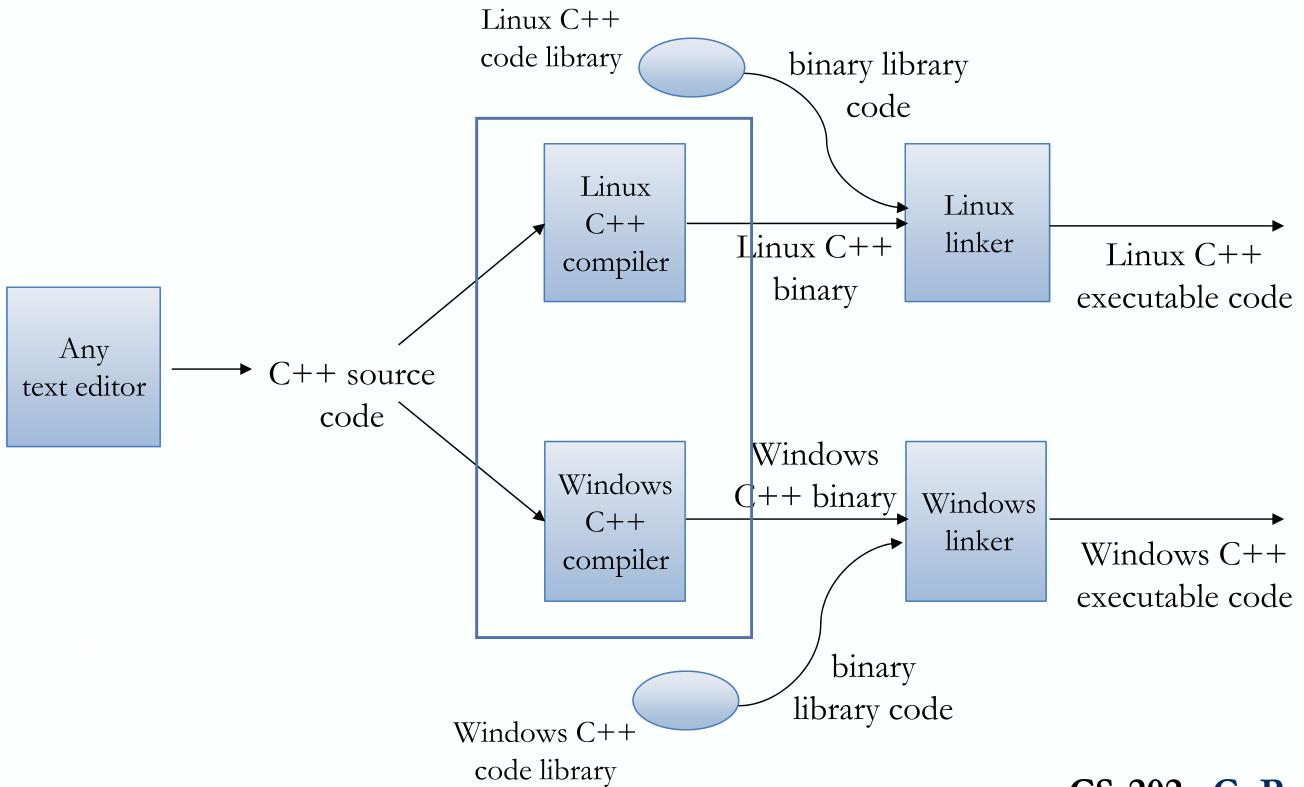
```
float quotient ;
                     3 / 4
```

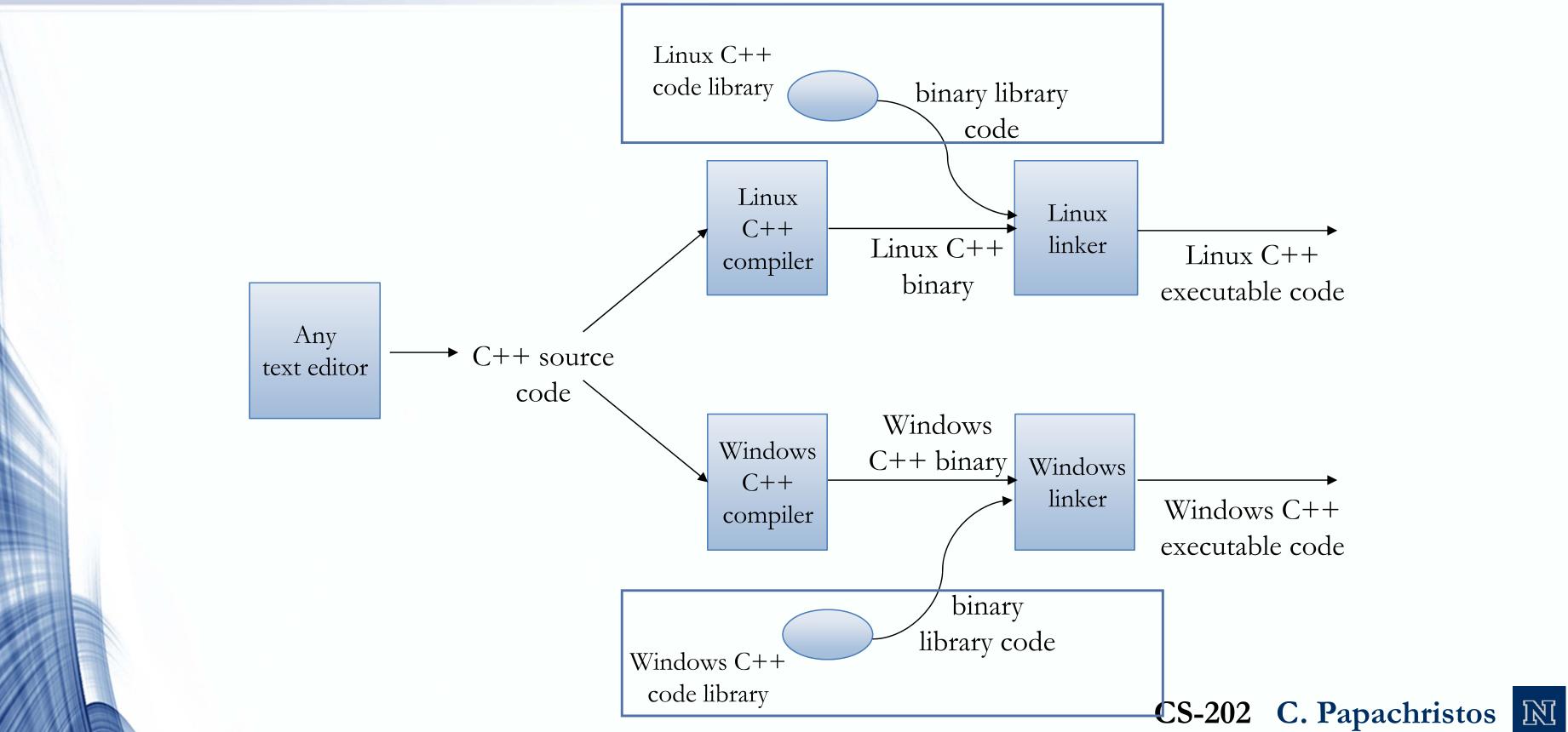
What's different about these two languages?

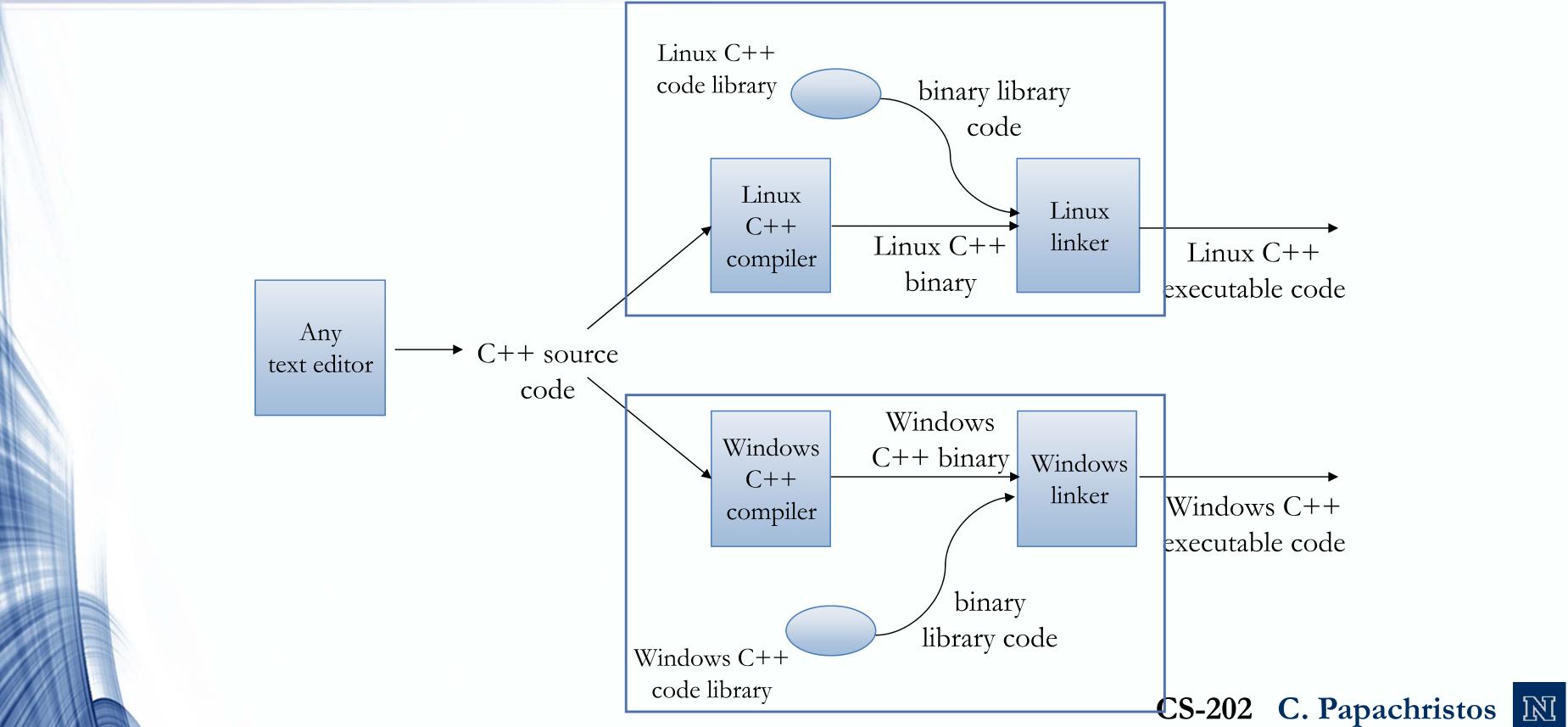
```
C++
#include <iostream>
using namespace std;
int main() {
 float quotient ;
  cout << "Hello, world" << endl ;</pre>
  quotient = 3 / 4;
  if (quotient == 0) {
   cout << "3/4 == 0";
   |cout << " in C++" ;
  } else {
    cout << "3/4 != 0";
  return 0;
```











http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

The C++ compilation process

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named prog1.cpp and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

- The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
- 2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
- 3. The assembler code generated by the compiler is assembled into the object code for the platform.
- 4. The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

By using appropriate compiler options, we can stop this process at any stage.

1. To stop the process after the preprocessor step, you can use the -E option:

```
g++ -Wall -std=c++11 -E prog1.cpp
```

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

2. To stop the process after the compile step, you can use the -s option:

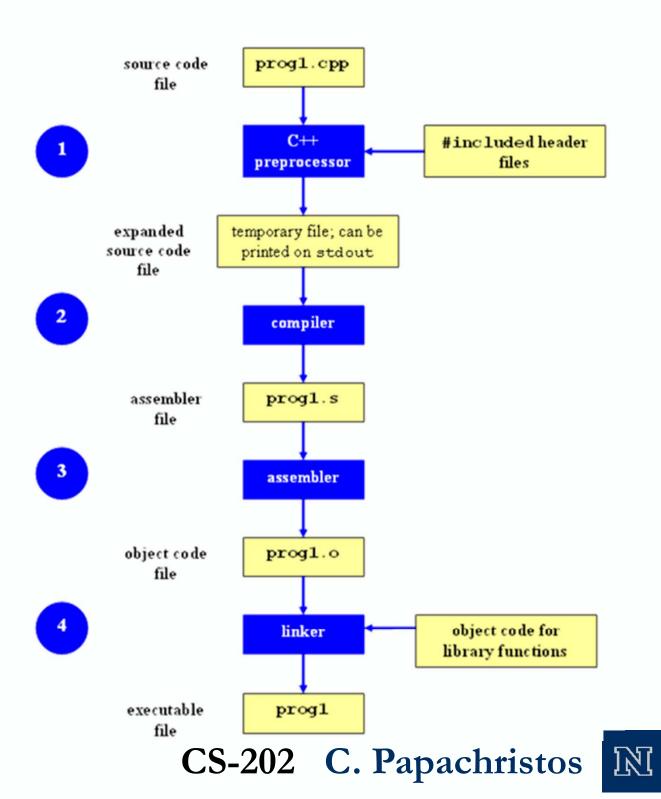
```
g++ -Wall -std=c++11 -S progl.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.s.

3. To stop the process after the assembly step, you can use the -c option:

```
g++ -Wall -std=c++11 -c prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.o.



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

The C++ compilation process

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named prog1.cpp and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

- The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
- 2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
- 3. The assembler code generated by the compiler is assembled into the object code for the platform.
- The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

By using appropriate compiler options, we can stop this process at any stage.

1. To stop the process after the preprocessor step, you can use the -E option:

```
g++ -Wall -std=c++11 -E prog1.cpp
```

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

2. To stop the process after the compile step, you can use the -s option:

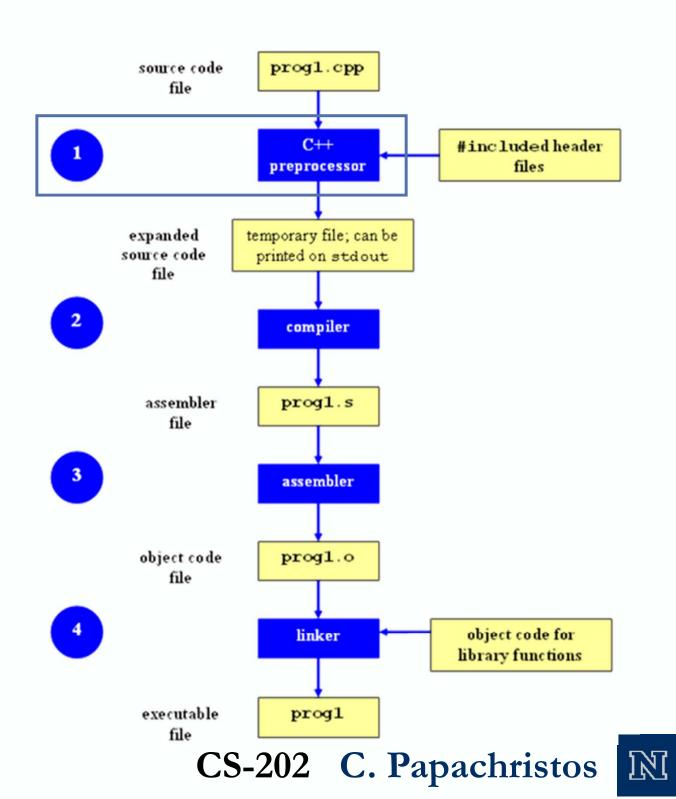
```
g++ -Wall -std=c++11 -S prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.s.

3. To stop the process after the assembly step, you can use the -c option:

```
g++ -Wall -std=c++11 -c prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.o.



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

The C++ compilation process

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named prog1.cpp and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

- 1. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
- 2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
- 3. The assembler code generated by the compiler is assembled into the object code for the platform.
- The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

By using appropriate compiler options, we can stop this process at any stage.

1. To stop the process after the preprocessor step, you can use the -E option:

```
g++ -Wall -std=c++11 -E prog1.cpp
```

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

2. To stop the process after the compile step, you can use the -s option:

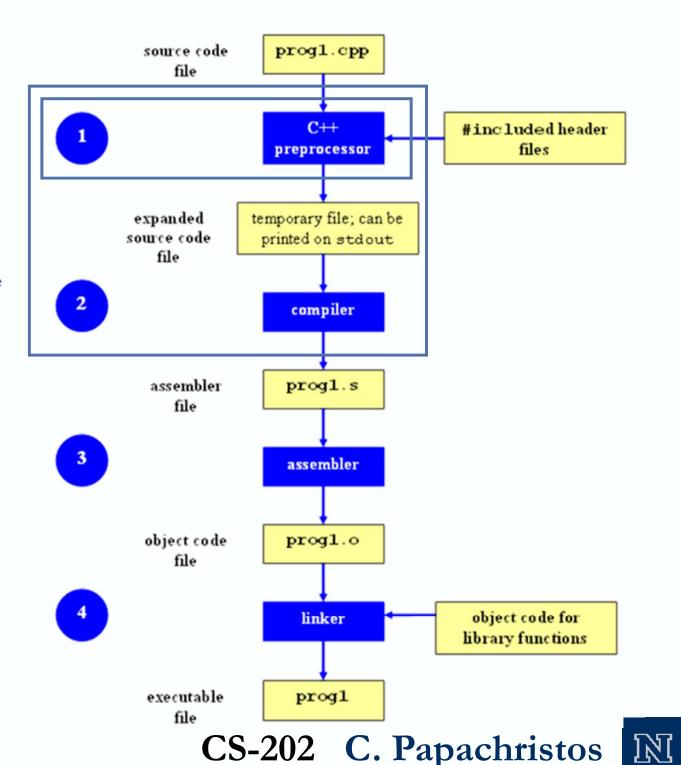
```
g++ -Wall -std=c++11 -S progl.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.s.

3. To stop the process after the assembly step, you can use the -c option:

```
g++ -Wall -std=c++11 -c prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.o.



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

The C++ compilation process

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named prog1.cpp and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

- 1. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
- 2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
- 3. The assembler code generated by the compiler is assembled into the object code for the platform.
- 4. The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

By using appropriate compiler options, we can stop this process at any stage.

1. To stop the process after the preprocessor step, you can use the -E option:

```
g++ -Wall -std=c++11 -E prog1.cpp
```

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

2. To stop the process after the compile step, you can use the -s option:

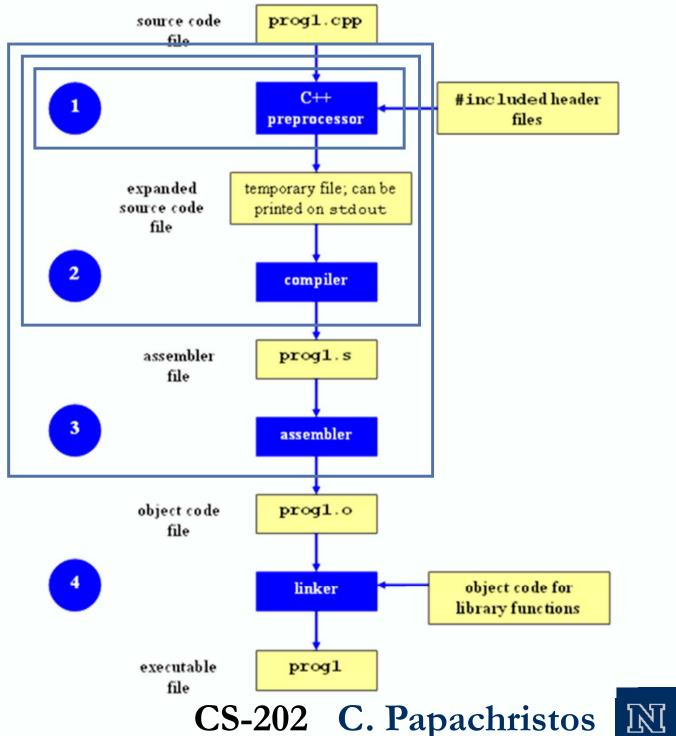
```
g++ -Wall -std=c++11 -S prog1.cpp
```

By default, the assembler code for a source file named *filename.cpp* will be placed in a file named *filename.s*.

3. To stop the process after the assembly step, you can use the -c option:

```
g++ -Wall -std=c++11 -c prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.o.



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

The C++ compilation process

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named prog1.cpp and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

- 1. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
- 2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
- 3. The assembler code generated by the compiler is assembled into the object code for the platform.
- 4. The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

By using appropriate compiler options, we can stop this process at any stage.

1. To stop the process after the preprocessor step, you can use the -E option:

```
g++ -Wall -std=c++11 -E prog1.cpp
```

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

2. To stop the process after the compile step, you can use the -s option:

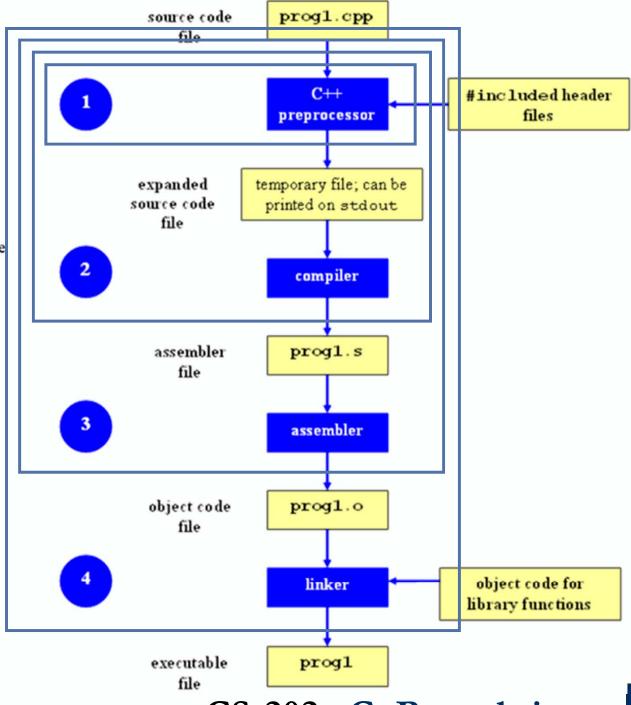
```
g++ -Wall -std=c++11 -S prog1.cpp
```

By default, the assembler code for a source file named *filename.cpp* will be placed in a file named *filename.s*.

3. To stop the process after the assembly step, you can use the -c option:

```
g++ -Wall -std=c++11 -c prog1.cpp
```

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.o.



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp
```

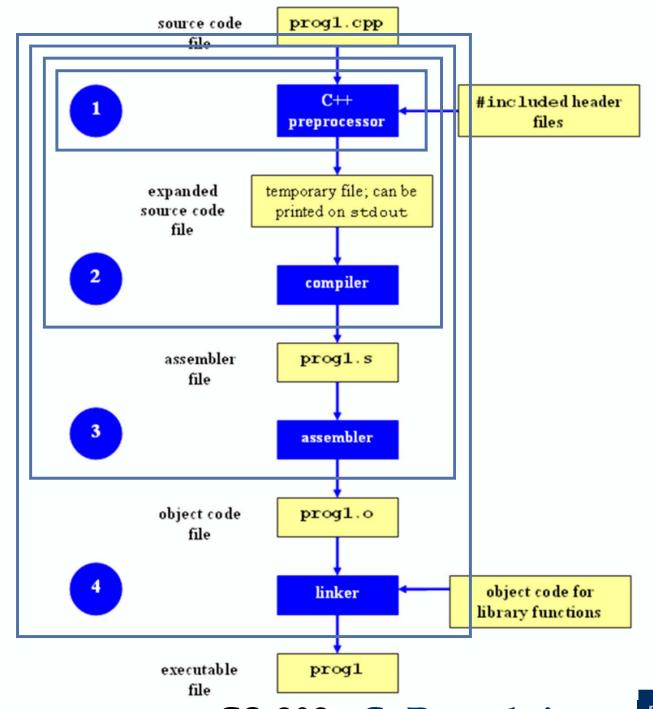
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile g++ -Wall -03 -std=c++11 -0 prog1 prog1.cpp

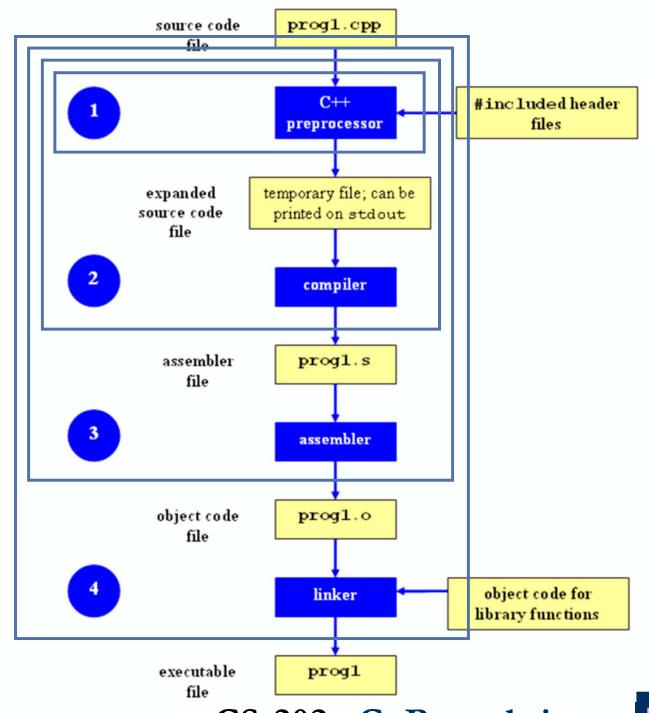
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp
```

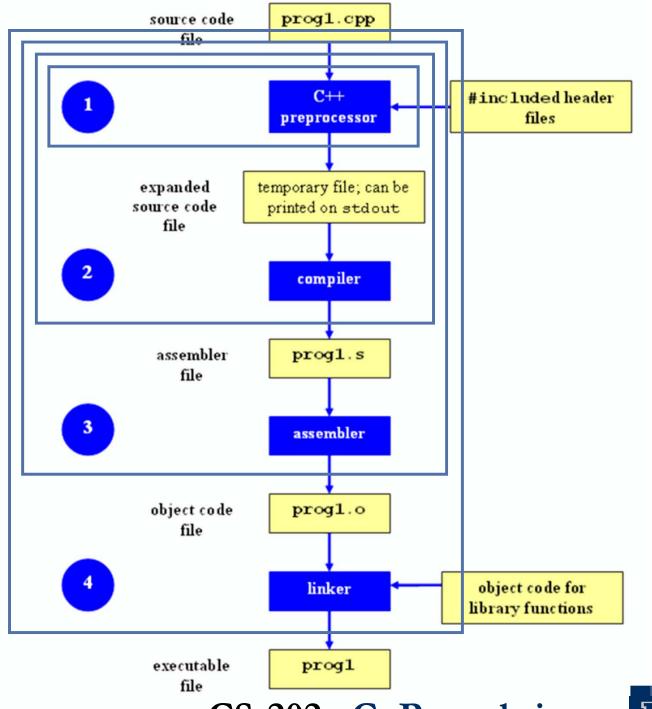
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

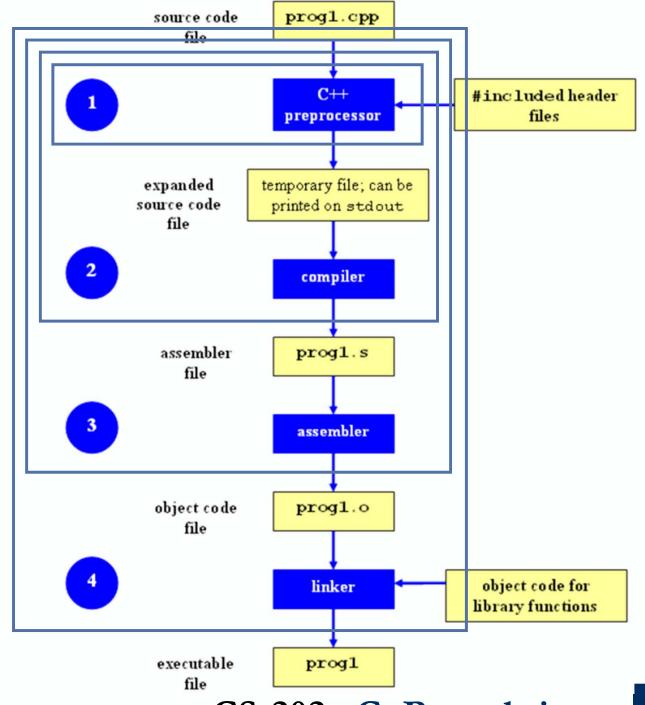
```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp
```

Compile Objects & Link g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)
./prog1



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

```
g++ -Wall -03 -std=c++11 -o prog1 prog1.cpp
```

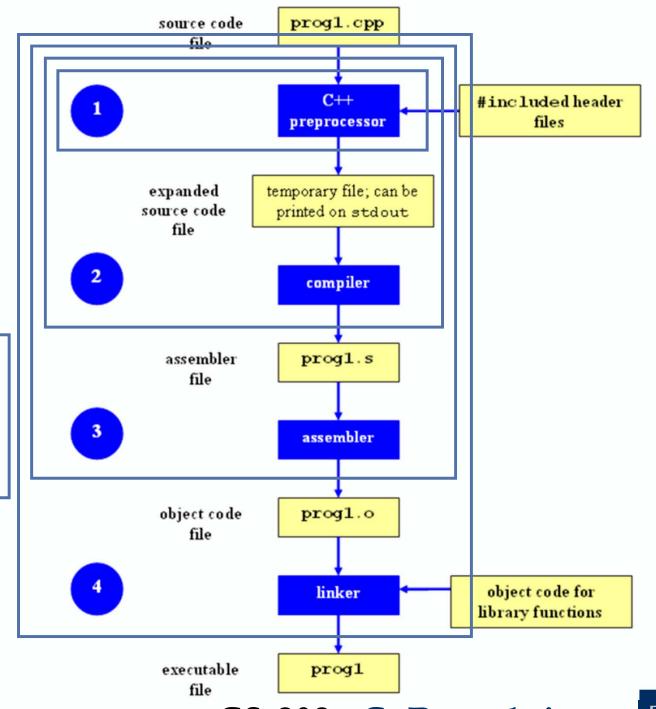
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)



CS-202 C. Papachristos



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

```
g++ -Wall -03 -std=c++11 -o prog1 prog1.cpp
```

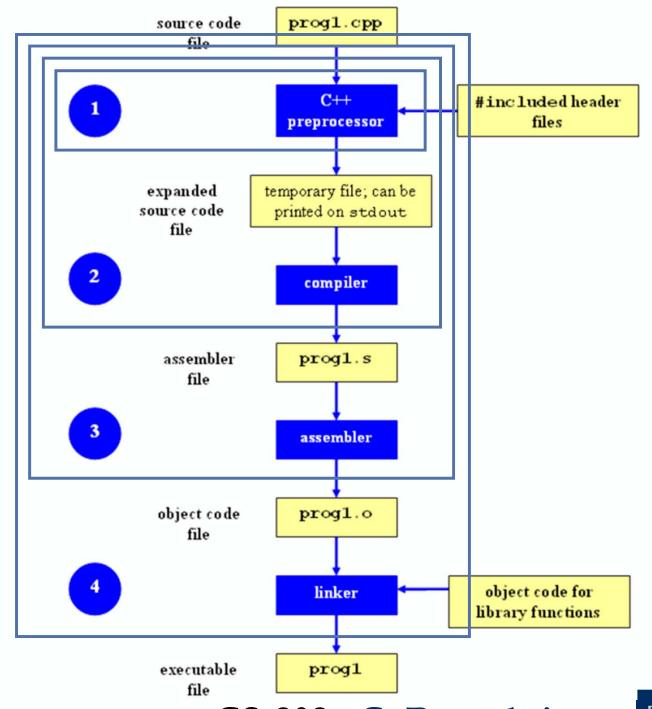
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)
./prog1



http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Standalone Compile

```
g++ -Wall -03 -std=c++11 -o prog1 prog1.cpp
```

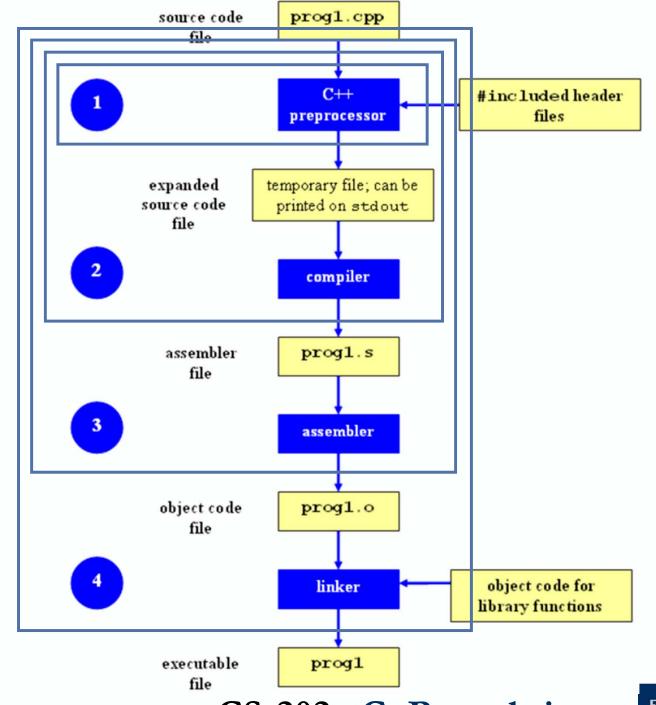
Compile Objects & Link

```
g++ -Wall -O3 -std=c++11 -c -o prog1.o prog1.cpp
g++ -Wall -O3 -std=c++11 -c -o prog2.o prog2.cpp
```

Linking

```
g++ -Wall -O3 -std=c++11 -o prog1 prog1.cpp -lm
g++ -Wall -O3 -std=c++11 -o prog12 prog1.o prog2.o -lm
```

Running under (Linux)
./prog1



```
#include <iostream>
    using namespace std;
    int main( )
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
10
         if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                   << "a more elementary book by the same author.\n";
12
         else
13
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
                            Includes, function Definitions, Preprocessor directives
    using namespace std;
                                  ++namespaces
    int main( )
        int numberOfLanguages;
        cout << "Hello reader.\n"</pre>
              << "Welcome to C++.\n";
        cout << "How many programming languages have you used? ";</pre>
        cin >> numberOfLanguages;
10
        if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                  << "a more elementary book by the same author.\n";
12
13
        else
            cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
    using namespace std;
                      The main
    int main( )
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
10
         if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                   << "a more elementary book by the same author.\n";
12
         else
13
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
    using namespace std;
    int main( )
                                     Variable Definition
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
10
         if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                   << "a more elementary book by the same author.\n";
12
         else
13
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
    using namespace std;
    int main( )
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
                                           Console Output
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
10
         if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                  << "a more elementary book by the same author.\n";
12
         else
13
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
    using namespace std;
    int main( )
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
                                       Console Input
         if (numberOfLanguages < 1)</pre>
10
             cout << "Read the preface. You may prefer\n"</pre>
11
                   << "a more elementary book by the same author.\n";
12
         else
13
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

```
#include <iostream>
    using namespace std;
    int main( )
        int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
10
         if (numberOfLanguages < 1)</pre>
             cout << "Read the preface. You may prefer\n"</pre>
11
                                                                           Selection Structures /
                  << "a more elementary book by the same author.\n";
12
                                                                           Flow Control
13
         else
             cout << "Enjoy the book.\n";</pre>
14
15
         return 0;
16
```

SAMPLE DIALOGUE I

Hello reader.

Welcome to C++.

How many programming languages have you used? 0 - User types in 0 on the keyboard.

Read the preface. You may prefer

a more elementary book by the same author.

SAMPLE DIALOGUE 2

Hello reader.

Welcome to C++.

How many programming languages have you used? 1 — User types in 1 on the keyboard. Enjoy the book

```
#include <iostream>
    using namespace std;
    int main( )
         int numberOfLanguages;
         cout << "Hello reader.\n"</pre>
 6
              << "Welcome to C++.\n";
         cout << "How many programming languages have you used? ";</pre>
         cin >> numberOfLanguages;
         if (numberOfLanguages < 1)</pre>
10
             cout << "Read the preface. You may prefer\n"</pre>
11
12
                   << "a more elementary book by the same author.\n";
13
         else
             cout << "Enjoy the book.\n";</pre>
14
                       Output
         return 0;
15
16
```

C++ Identifiers & Variables

C++ Identifiers

Can't use keywords/reserved words.

Case-sensitivity and validity of identifiers.

Meaningful names!

Used for variables, class names, and more.

Variables

A memory location to store data for a program.

Must declare all data before use in program.

Variable Declaration

C++ Variables

When we declare a variable, we tell the compiler:

When and where to set aside memory space for the variable.

How much memory to set aside.

How to interpret the contents of that memory;

I.e., the specified data type.

What name we will be referring to that location in memory;

I.e. by its identifier, or name.

Variable Declaration

C++ Variables

```
Syntax: <type> <legal identifier> ;
Examples:
   int sum ;
   float average ;
   double grade = 98 ;
```

Must be declared before being used

Must be declared to be of a given type (e.g. int, float, char, etc.)

Don't forget the semicolon at the end!

Variable Declaration

C++ Variables

Naming conventions are rules for names of variables to improve readability

Different standards exist, suggested:

Start with a lowercase letter

Indicate "word" boundaries with an uppercase letter

Restrict the remaining characters to digits and lowercase letters

topSpeed bankRate1 timeOfArrival

Note: variable names are still case sensitive!

Most common standard:

Google standard https://google.github.io/styleguide/cppguide.html

Data Types

Simple Types

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
short (also called short int)	2 bytes	-32,768 to 32,767	Not applicable
int	4 bytes	-2,147,483,648 to 2,147,483,647	Not applicable
long (also called long int)	4 bytes	-2,147,483,648 to 2,147,483,647	Not applicable
float	4 bytes	approximately 10 ⁻³⁸ to 10 ³⁸	7 digits
double	8 bytes	approximately 10 ⁻³⁰⁸ to 10 ³⁰⁸	15 digits

long double	10 bytes	approximately 10 ⁻⁴⁹³² to 10 ⁴⁹³²	19 digits
char	ı byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
bool	ı byte	true, false	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. Precision refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types float, double, and long double are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Data Types

Simple Types

	TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
	short (also called short int)	2 bytes	-32,768 to 32,767	Not applicable
	int	4 bytes	-2,147,483,648 to 2,147,483,647	Not applicable
	long (also called long int)	4 bytes	-2,147,483,648 to 2,147,483,647	Not applicable
	float	4 bytes	approximately 10 ⁻³⁸ to 10 ³⁸	7 digits
>	double	8 bytes	approximately 10 ⁻³⁰⁸ to 10 ³⁰⁸	15 digits

long double	10 bytes	approximately 10 ⁻⁴⁹³² to 10 ⁴⁹³²	19 digits
char	ı byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
bool	ı byte	true, false	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. Precision refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types float, double, and long double are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Mostly seen/used simple types.

Data Types

Data Type handling

One of the big changes from Python to C++.

Variables can only be of one type

- A string cannot be changed into a list
- A tuple cannot be changed into a dictionary
- An integer is always an integer forever
- A variable's type must be explicitly declared

Data Assignment

You can initialize data in declaration statement Results will be "undefined" if you don't initialize! int myValue = 0;

Assigning data during execution

Lvalues (left-side) & Rvalues (right-side)

Lvalues must be variables

Rvalues can be <u>any expression</u>

```
Example: distance = rate * time;

Lvalue: "distance"

Rvalue: "rate * time"
```

Data Assignment

Compatibility of Data Assignments

Type mismatches

Cannot place value of one type into variable of another type

Only the integer part "fits", so that's all that goes

Called "implicit" or "automatic type conversion"

Literals

2, 5.75, 'Z', "Hello World\n"

Also known as "constants": can't change in program

Program Data

Literals / Literal Data

```
Examples:
                    // Literal constant int
  5.75
                    // Literal constant double
   'Z'
                 // Literal constant char
```

"Hello World\n" // Literal constant string

Cannot change values during execution

Called "literals" because you "literally typed" them in your program!

Program Data

Constants / Constant Data

You should not use literal constants directly in your code – It might seem obvious to you, but not so:

Is this weeks per year... or cards in a deck?

- Instead, use named constants.
- > Give the constants names.

Also allows you to change multiple instances in a central place.

Program Data

Constants / Constant Data

Two ways to go about this:

One way: Preprocessor Define #define WEEKS PER YEAR 52 (Note: there is no "=")

Alternative way: Constant Variable Just add the keyword "const" to the declaration const float PI = 3.14159;

CS-202 Time for Questions! CS-202 C. Papachristos