COSC 220: Data Structures in C++

Lecture 1

Early days programs written in Machine Language
i.e. primitive instructions executed directly by machine.
Language based on hardware → hard to follow

1955 intro Fortran language: ex. of higher level language

C programming language: Bell Labs 1972

C
PROGRAMMING

COSC 220: Data Structures in C++

Lecture 1

C++ → Object oriented C language
C++ includes C as a subset language. Intro'd in 1983.

C++ has evolved: revised in 1998, 2003, 2011, and rec'ly 2014.
C++11 standard since most compilers don't support C++14 yet

COSC 220: Data Structures in C++

Lecture 1

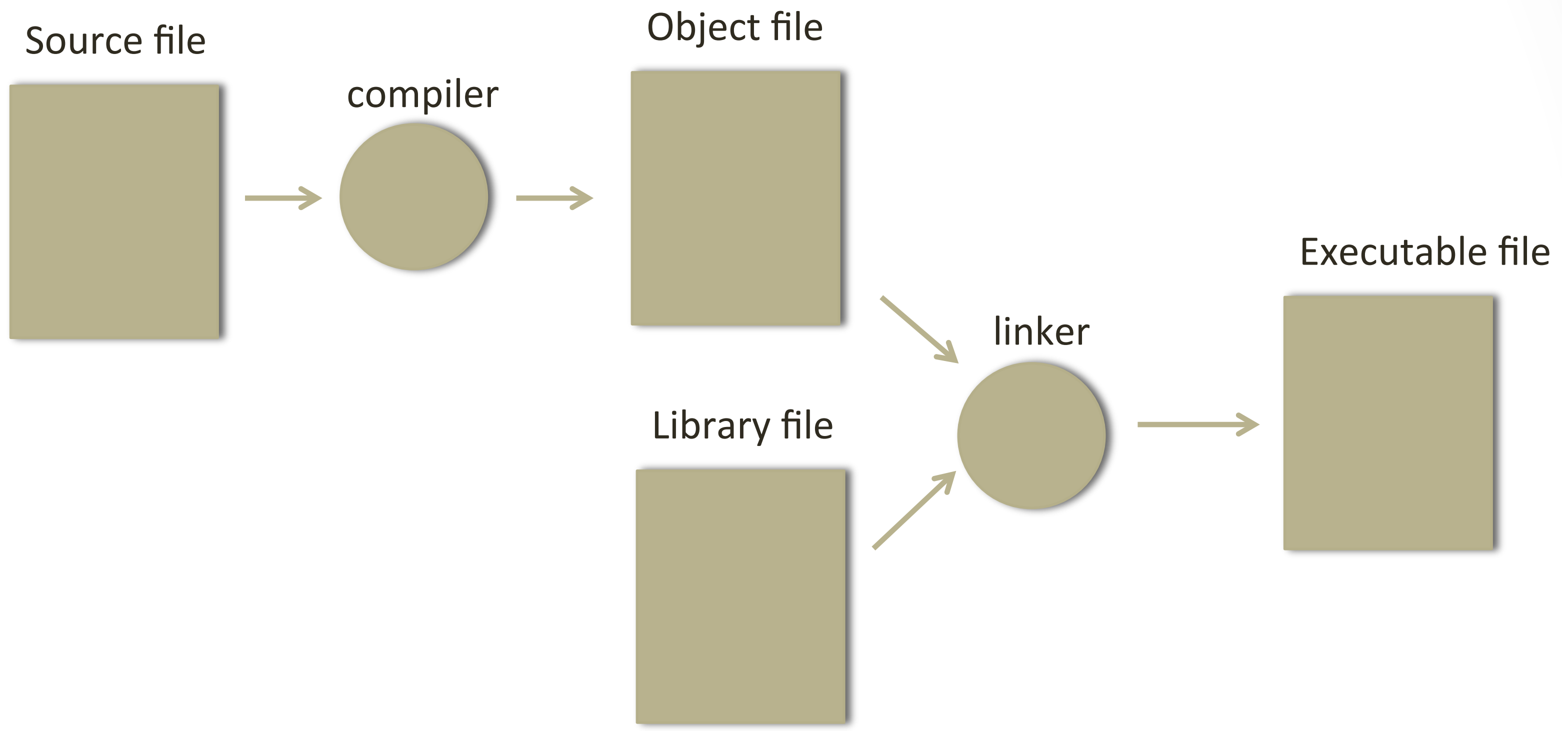
C++: compiled language
Python: scripting language
Matlab: integrated IDE + scripting language

Scripting Lang. → fast prototyping i.e. R&D
Compiled Lang. → fast running after compiling

COSC 220: Data Structures in C++

Lecture 1

Compilation Process:



1. **Source file**: text of your code
2. **Compiler** → translate source to **object file**
3. **Object file**: machine-language instructs
4. Object file combined with other object files via Linker → **Executable file**
5. Other object files: predefined obj files called **libraries**
6. **Libraries**: contain machine-lang instructs for various operations req'd by program
7. **Linking**: combining object files into an executable

First C++ program

```
/*
* hello world program
*/

// first program in all programming courses

#include <iostream>
using namespace std;

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

COSC 220: Data Structures in C++

Lecture 1

First C++ program

```
/*
* hello world program
*/

// first program in all programming courses

#include <iostream>
using namespace std;

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

/*...*/ ← multiline comment

// ← single line comment

← #include <library>, iostream library

← using namespace std, create namespace where standard functions & other data come from

← all C++ code needs main function

← braces needed instead of tabbing (python)

← cout: output. "..." indicates string. end line = endl

← return value 0 meaning code ran w/o issue

note: semicolons needed after each statement

Pretty complicated for such a simple program.

COSC 220: Lecture 2

COSC 220: Data Structures in C++

Lecture 2

First C++ program

```
/*
* hello world program
*/

// first program in all programming courses

#include <iostream>
using namespace std;

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

/*...*/ ← multiline comment

// ← single line comment

← #include <library>, iostream library
← using namespace std, create namespace where standard functions & other data come from
← all C++ code needs main function
← braces needed instead of tabbing (python)
← cout: output. "..." indicates string. end line = endl
← return value 0 meaning code ran w/o issue
note: semicolons needed after each statement

Pretty complicated for such a simple program.

COSC 220: Data Structures in C++

Lecture 2

Unix like systems like OSX, Linux, Cygwin you can compile and run C++ from the command line

- example of compiling and running hello.cpp from command line (in class demo)
- example of using emacs to edit, compile, and run from the command line in OSX (terminal) (in class demo)
- Note: CLion also has the command line option (in class demo)

COSC 220: Data Structures in C++

Lecture 2

So far, C++ seems overly complicated for such simple things...
So why use C++ over other languages?

Main reason: Speed!!!

All a balancing act:
Prototyping/R&D time vs *Implementation time*

Quantitative Finance, Apple IOS/OS applications, Sony, etc. etc. need speed. Most code is in C++.
Initial rapid prototyping often done using Matlab or Python.

Main Point: cannot always say one is better, it depends on what you are trying to do!

COSC 220: Data Structures in C++

Lecture 2

Download:
- speed_test.cpp (C++)
- speed_test.py (Python)
- speed_test.m (Matlab)

from www.fredpark.com/teaching

COSC 220: Data Structures in C++

Lecture 2

Download:

- speed_test.cpp (C++)
- speed_test.py (Python)
- speed_test.m (Matlab)

from www.fredpark.com/teaching

100,000,000 iterations adding i + i^2 + i^3, i=1:N
C++: ~0.85 secs
Python: ~90 secs
Matlab: ~ 13 secs

C++ is 100x faster than Python and 15x faster than Matlab
Difference even more noticeable as data gets larger and data structures are used

COSC 220: Data Structures in C++

Lecture 2

powers of 2 code: see website

load and run

COSC 220: Lecture 3

Dissecting the powers of two code:

```
/* File: PowersOfTwo.cpp
 * This program generates a list of the powers of
 * two up to an exponent limit entered by the user.
 */

#include <iostream> //iostream library
using namespace std;

/* Function Prototypes */
int raiseToPower(int n, int k);

/* Main Program */
int main() {
   int limit;
   cout << "This program lists powers of two." << endl;
   cout << "Enter exponent limit: ";
   cin >> limit;
   for (int i = 0; i <= limit; i++) {
      cout << "2 to the " << i << " = "
          << raiseToPower(2, i) << endl;
   }
   return 0;
}

// Function raiseToPower
// Usage: int p = raiseToPower(n, k);
//Returns the integer n raised to the kth power.

int raiseToPower(int n, int k) {
   int result = 1;
   for (int i = 0; i < k; i++) {
      result *= n;
   }
   return result;
}
```

COSC 220: Lecture 4

Dissecting the powers of two code:

```
/* File: PowersOfTwo.cpp
 * This program generates a list of the powers of
 * two up to an exponent limit entered by the user.
 */
```

/* ... */ ← multi-line comments
// ← single line comments

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

#include <iostream>: example of a library.
libraries: collections of prev'ly written tools that perform useful operations

iostream library: important input output operations based on data structure called stream

#include <iostream>: statement tells compiler read relevant def's from a header file.
<std c++ system library> between '<>' brackets

Dissecting the powers of two code:

```
/* Main Program */
int main() {
....
return 0;
}
```

1. all C++ programs have a main function
2. ‘int main()’ ← variable declaration. reserves space for value used by program
3. ‘return 0’ ← value returned by main. ‘0’ means program ran successfully
4. indentations not syntactically meaningful in C++. Must use brackets: ‘{...}’ to indicate body of the function
5. Indents in Python are syntactically meaningful (cf. COSC 120)

Dissecting the powers of two code:

```
int limit;
   cout << "This program lists powers of two." << endl;
   cout << "Enter exponent limit: ";
   cin >> limit;
```

1. `int limit’ ← variable declaration. note: scope is to the end of the block of which it is defined. example of local variable.
2. ‘cout’ ← console output stream def’d by iostream.
3. ‘endl’ ← end line, also output by ‘cout’
4. ‘cout << "Enter exponent limit: ”;’ ← prompt
5. ‘cin>>limit’ ← console input stream, places input into ‘limit’ variable
6. note: ‘limit’ is of type int. ‘cin’ auto converts input to type int.

Dissecting the powers of two code:

For Statement:

```
for (int i = 0; i <= limit; i++) {
        cout << "2 to the " << i << " = "
             << raiseToPower(2, i) << endl;
   }
```

} header line

} loop body

1. For statement: contains header line and loop body
2. includes a function call in the loop body
3. note: statements can be split across lines. see loop body.
4. function call 'raiseToPower(2, i)' suspends execution of main function until a value is returned by 'raiseToPower(2, i)'

Q: How are function definitions handled in C++?

Dissecting the powers of two code: Function Definitions and Prototypes:

```
/* Function Prototypes */
int raiseToPower(int n, int k);

 << raiseToPower(2, i) << endl;

// Function raiseToPower
// Usage: int p = raiseToPower(n, k);
//Returns the integer n raised to the kth power.

int raiseToPower(int n, int k) {
   int result = 1;
   for (int i = 0; i < k; i++) {
      result *= n;
   }
   return result;
}
```

← function prototype

← function call in main

← function definition
(after main())

function def'n:

1. prototype before main(), tells compiler how to call the fctn.
2. must provide declaration or def'n of fctn before making any calls to it.

Variable Declarations

variables: name and type needed

- syntax: *type namelist;*
- placement det's scope: region in code where it's accessible. Usually to end of block where it is def'd.
- multiple variables defined as follows:

```
        double n1, n2, n3;
```

- run: 'AddThreeNumbers.cpp' code from my webpage
- can define & initialize at same time:

```
        int sum = 0;
```

- initial value called an *initializer*

See book for naming conventions
Cannot be a reserved word

Variable Declarations

Reserved words in C++

auto const double float int short struct unsigned break continue else for long signed switch void case default enum goto register sizeof typedef volatile char do extern if return static union while

asm dynamic_cast namespace reinterpret_cast try bool explicit new static_cast typeid catch false operator template typename class friend private this using const_cast inline public throw virtual delete mutable protected true wchar_t

C++ 11 adds:

and bitand compl not_eq or_eq xor_eq
and_eq bitor not or xor

Local and Global Variables

1. local and global variables
2. constants
3. data types, int, bool, float, double etc.
4. expressions and operators avail in C++
5. type conversion hierarchy for numeric types
6. int division, remainder, and typecast
7. increment, decrement and boolean operators
8. statements
9. ex: AddIntegerList.cpp
10. 1-9 more readily apparent from exercises

Local and Global Variables

1. local and global variables
2. constants

Scope of local variable is to end of code block at which it is declared
Scope of global variable is remainder of the file in which it's declared
ex:

```
int spare = 1; // global variable

main() {
double sum = 0; //local variable
}
```

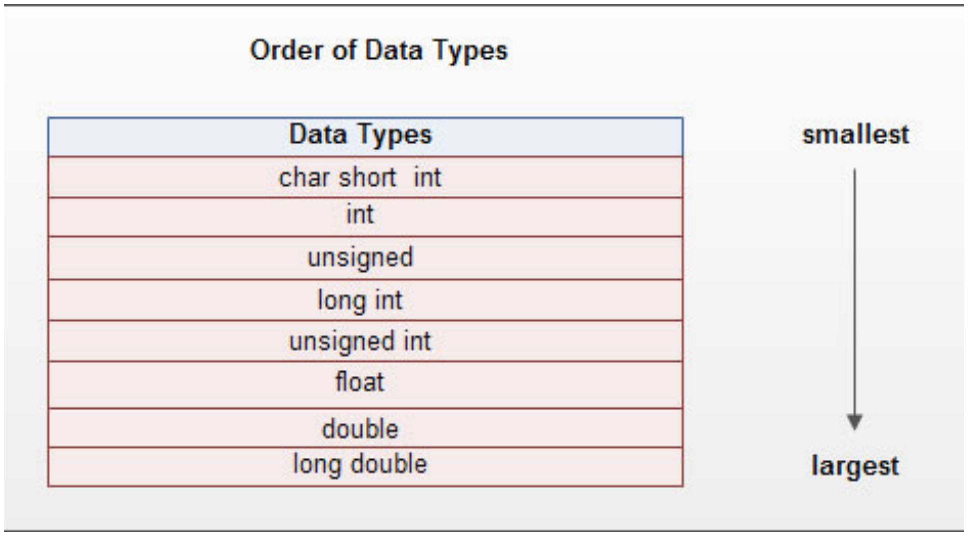
Avoid global variables at all costs... except: constants!

```
const double pi = 3.141592653;
main() {
double sum = 0; //local variable
}
```

data types

1. data types, int, bool, float, double etc. see code.
2. expressions and operators avail in C++. see code
3. type conversion hierarchy for numeric types
4. int division, remainder, and typecast
5. increment, decrement and boolean operators
6. statements
7. ex: AddIntegerList.cpp

1-6 more readily apparent from class exercises



Order of Data Types

Data Types	
char short int	smallest
int	
unsigned	
long int	
unsigned int	
float	
double	
long double	largest

COSC 220: Data Structures in C++

Lecture 4

Class exercises:

1. prompt the user to enter an integer N and print “COSC 220 is Awesome” N-times.
2. Write code that prompts the user to enter a list of numbers using a sentinel to stop the input. Then compute the average of them and print them out.
3. Write code to calculate N! where you must use a function other than main in your program. Do it by using an increment operator in a loop and also a decrement operator.
4. Approximate pi by calculating the first 10,000 terms of the following series and multiplying it by 4:
pi approx = 4*(1 − 1/3 + 1/5 -1/7 + 1/9 − 1/11+ ...)
5. Write code that reverses an integer that is entered:
i.e. 12345678 →87654321
6. Write code that reads a list of integers and outputs the largest value right after the sentinel appears.
7. Write code that outputs the largest & second largest values in list of numbers entered prior to sentinel being entered. Output after sentinel is entered.
8. Approximate the area of a quarter circle by using 10,000 rectangles.