If learning C is like learning “business English,” learning C++ is like learning the rest of English.
Sometimes programming in C is the best or only option

- You “inherited” C code
- No C++ compiler is available for system you’re targeting
- Your software must work closely with the Linux kernel, or other C-based software

If we started a new project today, especially if it was big or involved many people, we’d probably choose C++
Classes – like Java classes
Templates – like Java generics
Standard Template Library – like java.util
More convenient text input & output
C++ is not a “superset” of C; many C programs don’t immediately work in C++

Think of C and C++ as closely related but different languages
Most concepts and constructs in C work *the same way* in C++:

- **Types:** int, char, char *, etc
  - C++ adds bool (equals either true or false)
- **Numeric representations & properties**
- **Arrays, pointers, * and &**, pointer arithmetic
- **if/else if/else, switch/case, for, while, do/while**
- **Pass by value** (still the default), pass by pointer
- **Stack vs. heap, scope & lifetime**
- **Operators:** arithmetic, relational, logical, assignment, bitwise
- **struct** (minor differences)
- **Casting** (minor differences)
Our favorite tools work just as well with C++:

- git
- make
- gdb
- valgrind
#include <iostream>

using std::cout;
using std::endl;

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

$ g++ -c hello_world.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hello_world hello_world.o
$ ./hello_world
Hello world!
Programming stages same as for C: edit -> compile -> execute

When compiling:

- g++ instead of gcc
- -std=c++11 instead of -std=c99
- .cpp instead of .c

$ g++ -c hello_world.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hello_world hello_world.o
$ ./hello_world
Hello world!
Options we used with gcc work with g++ too

- `-o` to set name of executable
- `-c` to compile to `.o` file
- `-g` to include debug symbols
- `-Wall` `-Wextra` `-pedantic` for sensitive warnings & errors
#include <iostream>

As with C, C++ library headers are included with `< angle brackets >`

For standard C++ headers, *omit the trailing .h*

- `<iostream>`, not `<iostream.h>`

#include "linked_list.h"

User-defined headers use " quotes " and end with .h as usual

- You’ll sometimes see .hpp instead of .h, but we’ll use .h
Can use familiar C headers: assert.h, math.h, ctype.h, stdlib.h, ...

When #include'ing, drop .h & add c at the beginning:

```cpp
#include <iostream>
#include <cassert> // dropped .h, added c at beginning

using std::cout;
using std::endl;

int main(int argc, char *argv[]) {
    assert(argc > 1); // our old friend assert
    cout << "Hello " << argv[1] << "!" << endl;
    return 0;
}
```