# Introduction to C, C++, and Unix/Linux

CS 60

C++ Introduction

Today

→ Reading [KR] Chapters 1-7

→ Read [So] chapters 1, 3, 4 (Boolean), 9, 13, 14, 18, &10.

## Notes

Oualline book (Practical C++ Programming)

- You won't be assigned chapters 2, 4, 5, 6, 7, 8, 11, 12, and several others – but read them anyway! They will reinforce the material from [KR].

## From Chapter 3

"Contrary to popular belief, programmers do not spend most of their time writing programs. Far more time is spent maintaining, upgrading, and debugging existing code than is ever spent on creating new work."

Programs are for the machine (instructions what to do) **and** for the programmer (description of what it does)

Comment liberally (though succinctly) **and** write code that is clearly readable even without comments

- Obvious, English-like variable and function names
- Consistent naming style
- Good consistent formatting (indentation, braces, etc.)
- Modular structure (small functions)

## So far we've covered…

- Compiling with gcc
- Header files
- File structure (multiple files)
- Basic I/O
- Variables (local, global, static, const…) and scope
- Data types, type casting
- Structs, unions, typedef
- C operators, control flow, expressions and statements, functions
- Creating and using C libraries
- Arrays and pointers, pointer arithmetic
- Memory allocation
- C standard library functions

```
void func(int& x)
{
  x = 0; // Initialize
  /********* Function *************
  //       for Quiz
  /////////////////////////////////
  x = 2005; /* Change value */
  std::cout << “x is “ << 2*x;
}
... // in main
int z = 123;
std::cin >> z;
func(z);
```

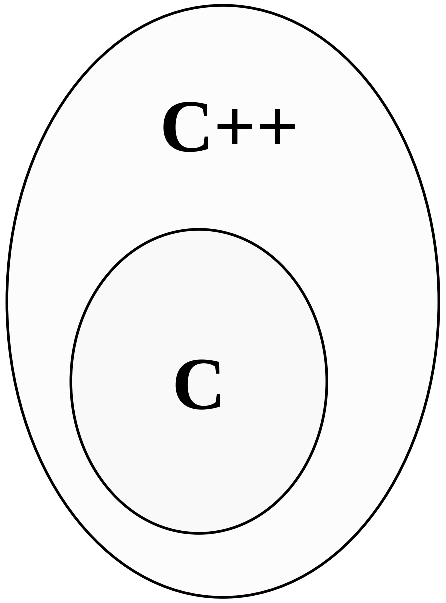
Commented

“**`x is 0`**”

**`z`** is 0

## What is C++?

- An <u>object-oriented</u> programming language
- An extension to C
  - Developed by Bjarne Stroustroup
  - Adds an object-oriented paradigm to C
- A superset of C
  - All C code can be compiled by C++ (sort of: **`g++ -x c`**)
  - For better or for worse… (stuck with legacy issues)



## C vs. C++

- C++ adds some minor improvements to C
  - A new I/O system
  - Reference variables
  - Function overloading
  - Inline functions
  - Default function parameters
- And brand new things
  - Objects, templates, built-in memory handling, exception handling, ...

## Major benefits of C++

- Similar to C, which is widely known and used
- Object-oriented design, can lead to code reuse
  - Groups data with related functions
- Good performance
- Flexibility (low-level to high-level)

## C++ vs. Java

- Both are object-oriented
- Similar features
  - Polymorphism
  - Inheritance
  - Data encapsulation

## C++ vs. Java (cont.)

- Differences
  - C++ is platform-dependent
  - Java bytecode is platform-independent
  - No operator overloading in Java
  - No garbage collection in C++
  - No multiple inheritance in Java
  - Compiler doesn’t enforce structure as much in C++

## C++ features

- Inheritance
  - Through both structs and classes
  - Allows for specialization
  - Promotes reuse
  - Requires programming skill!
    - ♦ Can be inefficient and difficult to understand

## C++ features (cont.)

- Data encapsulation
  - Protects data
    - Public, protected, and private data and functions, like Java
  - Separates API from implementation
  - Promotes code reuse

## C++ features (cont.)

- Polymorphism
  - Objects act differently based on their run-time type
  - Via function overloading (same name, different class)
    - Function called depends on the data type
  - Reduces complexity
    - Common interface to multiple functions
      - SetPixelGray8bit, SetPixelFloat32bit, SetPixelColor32bit...
      - now just SetPixel()

## Example: From C to C++

```
#include <stdio.h>
#define MAX_STR_LEN 256
int main(int argc, char **argv)
{//Assume ” is the same as “
  char name[MAX_STR_LEN];
  printf(“What’s your name? “);
  scanf(“%s”, name);
  printf(“Hi there, %s!\n”, name);
  return(0);
}
```

## Example: From C to C++

/usr/include/c++

```
#include <iostream>
#define MAX_STR_LEN 256
int main(int argc, char **argv)
{//Assume ” is the same as “
  char name[MAX_STR_LEN];
  printf(“What’s your name? “);
  scanf(“%s”, name);
  printf(“Hi there, %s!\n”, name);
  return(0);
}
```

## Example: From C to C++

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{//Assume ” is the same as “
  std::string name;
  printf(“What’s your name? “);
  scanf(“%s”, name);
  printf(“Hi there, %s!\n”, name);
  return(0);
}
```

## Example: From C to C++

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{//Assume ” is the same as “
  std::string name;
  std::cout << “What’s your name? “;
  scanf(“%s”, name);
  std::cout << “Hi there, “ << name << ”!\n”;
  return(0);
}
```

## Example: From C to C++

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{//Assume ″ is the same as “
  std::string name;
  std::cout << ″What’s your name? ″;
  std::cin >> name;
  std::cout << ″Hi there, ″ << name << ″!\n”;
  return(0);
}
```

## Example: From C to C++

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{//Assume ” is the same as “
  using namespace std;
  string name;
  cout << ”What’s your name? ”;
  cin >> name;
  cout << ”Hi there, ” << name << ”!” <<endl;
  return(0);
}
```

## First full C++ program

- C++ header files
- Namespace
- User-defined class
- string
- cout, cin

```
#include <iostream>
#include <string>

using namespace std;

class HiThere
{
public:
   void sayHello(void);
protected:
  string name;
};
```

```
void HiThere::sayHello(void)
{
  cout << “What’s your name? “;
  cin >> name;
  cout << “Hi there, “ << name
       << ”!” <<endl;
}

int main(int argc, char **argv)
{
  HiThere greeting;
  greeting.sayHello();
  return(0);
}
```

## Compiling C++ files

- Filename options:
  - .C
  - .cc
  - .c++
  - .cp
  - .cxx
  - .cpp

To compile:

```
% g++ -o hello hello.cpp
```

**`g++`** calls **`gcc`** with the default language set to C++, and automatically specifies linking with the C++ library

## What will this print?

```
int i(0);
```

Initialize variable value

```
std::cout << i++ << i++ << i++ << endl;
```

```
Output:
210
```

No spaces
2 then 1 then 0

## New C++ types

- **`bool`** type
  - **`true`** or **`false`**
  - Uses values 1 and 0, but the compiler can do type checking
  - Should be used in logic statements (rather than “zero” and “non-zero”)
- **`wchar_t`** type
  - Wide characters (4 bytes)
  - For character sets beyond ASCII
  - Used in most professional programming now

## New C++ types (cont.)

- The C++ library provides a powerful string package (**`#include <string>`**)
  - Manages storage for you! But there's overhead....
  - Several operators are defined on strings, e.g.:

    **`=  []  +  at()  length()  substr()`**

```
std::string name(“Joe”);
name += “ Smith”;
len = name.length();
```

## New C++ types (cont.)

- **`wstring`** – just like string, but uses wide characters (**`wchar_t`**) instead of **`char`**

```
std::wstring name(L“Joe”);
name += L“ Smith”;
len = name.length();
```

## C strings and C++ strings

- C strings can co-exist with C++ strings, but they are not interchangeable
  - Must do conversion
  - See Chapter 5

```
#include <cstring>
char name[64];
std::strcpy(name, “Joe Smith”);
```

```
#include <string>
#include <cstring>
string person(“John Doe”);
char name[64];
std::strcpy(name, person.c_str());
```

Accesses a C-like string

## Standard input, output, error

- **`std::cout`** for writing to stdout
- **`std::cin`** for reading from stdin
- **`std::cerr`** for writing to stderr
- **`std::getline ...`**