

W3101.002 Intro to C++

Joshua Reich

`reich@cs.columbia.edu`

Department of Computer Science
Columbia University

Session 2



Dot files

Operating systems hide information they need in many places.

One such place is in specially located hidden files.

`.profile` is run whenever you open a shell session

`.emacs` is run whenever you open an emacs session

Admin, etc.

- ▶ Class policies, expectations & grading, textbooks etc.
- ▶ Setting up CUNIX account
- ▶ UNIX
- ▶ emacs
- ▶ high-level code vs. assembler vs. machine code
- ▶ using `g++` (e.g. `g++ filename -o outputname`)



C++

- ▶ `#include` directives
- ▶ main functions & return statements
- ▶ code blocks (e.g. `{ }`)
- ▶ `cout <<`
- ▶ expression vs. statement
- ▶ scope operators (e.g. `std::`) and namespaces (`using`)
- ▶ `cin >>`
- ▶ operators & associativity
- ▶ buffers & flushing (e.g. `endl`)
- ▶ comments
 - ▶ `/* multi-line */`
 - ▶ `//single-line`
- ▶ datatypes: `string`, `int`

What is a Variable?

A *variable* is the identifier of an *object* that holds some type of value.

Some examples

- ▶ the TV station number to which your TV is currently set
`channel=9;`
- ▶ the name of your best friend
`best_friend = ``Billy Bob``;`
- ▶ the beverage in your hot cup
`hot_cup_beverage = new Beverage(chai_tea);`

Variable Basics

- ▶ Declaration / Definition : `string name;`
- ▶ Initialization : `name = ' ' josh' ' ;` or `name (' ' josh' ') ;`
 - ▶ builtin: no default, dangerous, so always initialize w/ safe value!
 - ▶ object: default value provided by constructor
 - ▶ initialization is NOT the same as assignment
- ▶ Lifetime / Scope
 - ▶ local variables
 - ▶ global variables
- ▶ illegal variable names - *C++ keywords*

CODE: `declaration.cc`, `scope.cc`

TEXT: *Accelerated C++ A.1*

What is a Datatype?

Every variable holds something, but what sort of something is it?

The variable's *type* or *datatype* tells us!

Some examples:

- ▶ `int` - an integer number
- ▶ `double` - a floating point number
- ▶ `string` - a string of text
- ▶ `bool` - a true/false value
- ▶ `char` - a character
- ▶ `Elephant` - a large mammalian creature
- ▶ `std::string::size_type` - the type which indicates the size of a string

CODE: `tobig.cc`

Builtins and Everything Else

The C++ core provides the language features always available. These are rather few and there is good reason why (portability, simplicity, flexibility).

C++ object types can be broken into three rough categories

- ▶ builtin
- ▶ standard
- ▶ custom made (i.e., made by you or me)

Datatype Basics

- ▶ casting
 - ▶ explicit casting (when applicable)
 - ▶ old notation: `i = (int)d;`
 - ▶ new notation: `i = static_cast<int>(d);`
don't forget the parens!
 - ▶ implicit casting (streams cast everything to type `string`)
- ▶ `typedef` - its just an alias, or nickname for the type
- ▶ `const`
- ▶ Interfaces - operators, member functions, etc.

CODE: `casting.cc`, `typedef.cc`, `const.cc`,
`interface.cc`

TEXT: *Accelerated C++*, section A.2

What is an Operator?

An *operator* is a tool that acts on something.
Not much different than a function, but each one has got a special symbol.

Some examples:

- ▶ +
- ▶ !
- ▶ <<
- ▶ ==
- ▶ |

Operator Basics

- ▶ types of operators (i.e. logical, arithmetic, stream)
- ▶ operator overloading / polymorphism
- ▶ operators in combination
- ▶ pitfall: = vs. ==

CODE: `logical.cc`

TEXT: *Accelerated C++*, section A.3

Operators Applied to Strings

- ▶ adding strings
- ▶ dot operator and member functions

CODE: `interface.cc`

What is a Container?

A *container* holds things (in C++ they all need to be the same type).

There is much we will learn about containers, however today we will just focus on one of the most useful containers, `vector`.

Vector Basics

- ▶ declaration: `vector<T> vec;`
- ▶ vector is a *template* (as are most containers)
- ▶ adding elmt's:
`vec.push_back(something_of_type_T);`
- ▶ referencing elmt's: `vec[i];`
- ▶ getting a vector's size: `vec.size`

CODE: `vector.cc`

More About Vectors

- ▶ general purpose algorithms `sort`
- ▶ `iterators`
- ▶ performance issues
- ▶ finding algorithm descriptions
- ▶ containers can be used like anything else - containers in containers
- ▶ pitfall: Don't forget `#include <vector>`!
- ▶ pitfall: Vectors need to be fully typed - `vector<vector>` won't work!

Other Container Types

- ▶ Sequences
 - ▶ C++ Vectors
 - ▶ C++ Lists
 - ▶ C++ Double-Ended Queues
- ▶ Container Adapters
 - ▶ C++ Stacks
 - ▶ C++ Queues
 - ▶ C++ Priority Queues
- ▶ Associative Containers
 - ▶ C++ Bitsets
 - ▶ C++ Maps
 - ▶ C++ Multimaps
 - ▶ C++ Sets
 - ▶ C++ Multisets

WEB: <http://www.cppreference.com/cppstl.html>



What is Control Flow?

control flow determines in what order instructions are executed and how many times.

A *loop* allows a piece of code to be repeated over, and over, and over...

A *if/else* statement allows one to execute statements conditionally.

if/else Basics

- ▶ `if`
- ▶ `else`
- ▶ `else if`
- ▶ be careful with nested `if/else` statements
- ▶ conditional operator `cond ? expr1 : expr2`
- ▶ *conditional* evaluation - using `||` and `&&` as tests
- ▶ `switch` - must evaluate to an `int` value

CODE: `if_else.cc`

Loop Basics

- ▶ while
- ▶ loop on input
 - ▶ some shortcut operators
 - ▶ increment ++, --
 - ▶ compound assignment (e.g +=)
 - ▶ break
 - ▶ continue
 - ▶ w/ cin
 - ▶ example: exponentiation
- ▶ for

CODE: `exponentiation.cc`, `bad_spacing.cc`,
`loops.cc`, `print_vector.cc`



Iterators

- ▶ an iterator points to something in a container
- ▶ iterators can be move around
- ▶ iterators can be dereferenced *
- ▶ example: using iterators in loops - another way to print a vector

CODE: `iterator.cc`, `vector_in_vector.cc`

A Quick Tutorial on Random Numbers

- ▶ generating a random number: `<cstdlib>`
- ▶ seeding random numbers
- ▶ using time as your seed: `<ctime>`
- ▶ the modulo operator: `|`

CODE: `if_alternates_and_random.cc`