



uOttawa

L'Université canadienne
Canada's university

Advanced Programming Concepts with C++ CSI2372 – Fall 2024

Mohammad Alnabhan
EECS

Université d'Ottawa | University of Ottawa



uOttawa.ca

C++

- **Object-oriented programming language**
 - Data abstraction (class concepts)
 - Operator overloading
- **C/C++ (and Objective C) together are (still) the de-facto standard (except for web-centric applications)**
- **Combines a high-level language with low-level features**
 - C++ is a *superset* of C
 - C is a functional programming language
- **Goals:**
 - Augment C with the notion of classes and inheritance
 - Keep the same performance as C
 - Keep the same applicability as C

Brief History of C/C++

1967-1980	Development of Unix by Ken Thompson, Denis Ritchie and others at Bell Labs
1969-1973	C by Denis Ritchie, Bell Labs Based on B written by Ken Thompson, most of Unix written in C
1984	C++ by Bjarne Stroustrup, Bell Labs Object oriented programming constructs were added to C
1998	C++ Standard ISO/IEC 14882 and revised in 2003 as ISO/IEC 14882:2003
2011	C++ Standard C++11 “C++0x”, ISO/IEC 14882:2011
2014	C++ Standard ISO/IEC 14882:2014
2017	C++ Standard ISO/IEC 14882:2017
2020	Next scheduled release



Bjarne Stroustrup

C++

- C++ has been derived from the well-known programming language C.
- The name C++ is related to the expression C++, which we can write in a C program to increment a variable C.
- C++ is a much younger language than C, its use is already widespread, and its popularity will no doubt increase considerably as a result of the excellent quality of popular compilers such as Turbo C++ from Borland.
- One of the attractive aspects of C++ is that it offers good facilities from Object-Oriented Programming (OOP), but, as a hybrid language, it also permits the traditional programming style, so that programmers can shift to OOP id and when they feel the need to do so.

C++

- In this regard, C++ differs from some 'purely' OO languages, such as: Smalltalk, Eiffel and Java.
- Viewed from the angle of many C programmers, C++ is simply 'a better C'.
- Besides the important: class concept, essential to OOP, there are many other points in C++ that are not available to C programmers. To mention just a few, related to functions: function overloading, inline functions, default arguments, type-safe linkage, and the very simple requirement that functions be declared before they are used.
- In C, the old practice of using undeclared functions is still allowed in order to keep many existing C programs valid; in C++ it is not.

- **Strengths**

- Low-level systems programming
- High-level systems programming
- Generic programming
- Embedded code
- High-performance programming
- Numeric/scientific computation
- Games Programming
- General application programming

- **Weaknesses**

- Legacy of C
- Insecurities
- Complexity
- No standard GUI library

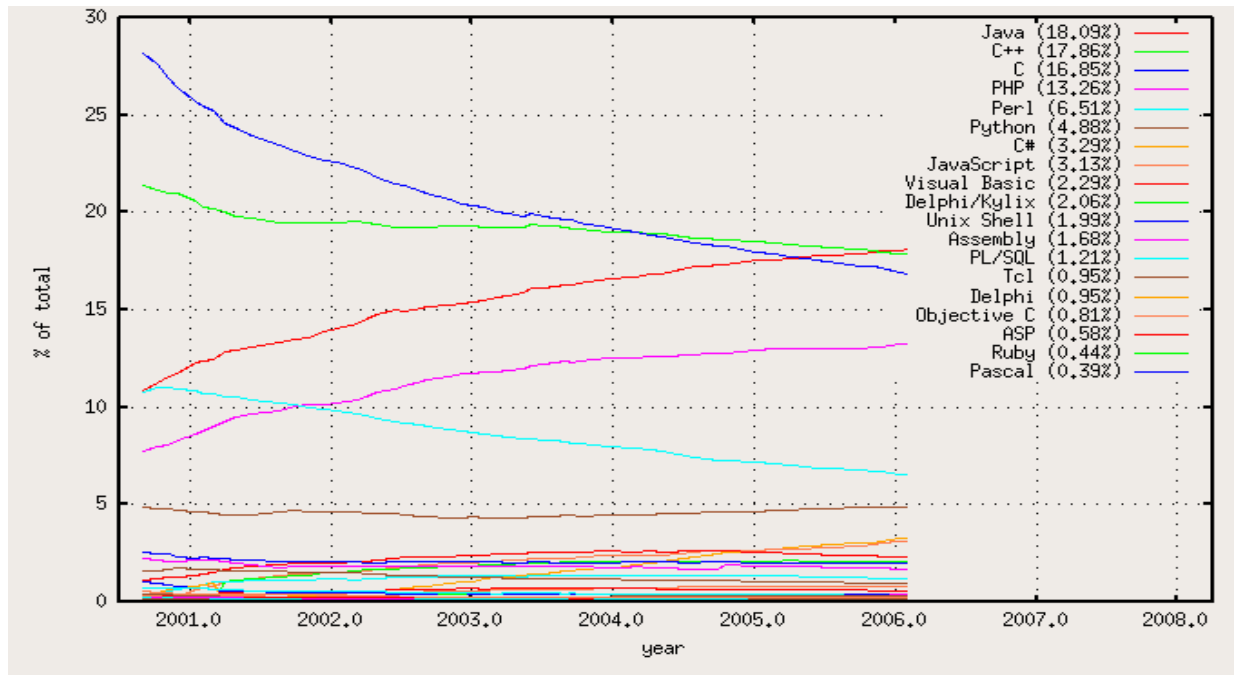
Popularity of Programming Languages



“The 2018 Top Programming Languages” IEEE Spectrum ranking [accessed Sep. 1, 2018]. Based on web searches, specific web pages, IEEE digital library etc.

Use of Programming Languages at the Beginning of the Century

- **François Labelle, Programming Language Usage Graph**
 - <https://wismuth.com/lang/languages.html>
 - Statistics based on open source projects at SourceForge



Is C++ in decline?

- **Bjarne Stroustrup:**

- *“No, I don't think so. C++ use appears to be declining in some areas and to be on an upswing in others. If I had to guess, I'd suspect a net decrease sometime during 2002-2004 and a net increase in 2005-2007 and again in 2010-2011, but I doubt anyone really knows. Most of the popular measures basically measures noise and ought to report their findings in decibel rather than "popularity." A professional survey in 2015 estimated the number of C++ programmers to be 4.4 million.”*
- See the Tiobe index at <https://www.tiobe.com/tiobe-index/> – a very popular measure
- *“There are more useful systems developed in languages deemed awful than in languages praised for being beautiful -- many more”*

Bjarne Stroustrup's FAQ: Did you really say that?. Retrieved on 2017-09-03.

Benefits of Learning C++

- Low-level control over many features including memory management, and, the breadth of C++
 - Improves understanding of software design
 - Helps to make informed choices about design
 - Bjarne Stroustrup: *"To use C++ well, you have to understand design and programming technique"* Bjarne Stroustrup's FAQ: Did you really say that?. Retrieved on 2017-09-03.
- Wide use and popularity of C/C++
 - Increases employment prospects
 - Helps to communicate with expert developers
 - Helps to evaluate and adapt projects by others

A First Look at C/C++

- Java syntax is based on C
- Execution of C/C++ starts with main
- System functions are not grouped in a class
- C++ has the concept of a **namespace**

Namespaces allow to group entities like classes, objects and functions under a name. This way the global scope can be divided in "sub-scopes", each one with its own name.

- **Example**

- Hello World in Java and C

- **Namespaces** defined:

- * Collection of name definitions

- For now: interested in namespace "std"

- * Has all standard library definitions we need

- **Examples:**

- ```
#include <iostream>
```

- ```
using namespace std;
```

- * Includes entire standard library of name definitions

- ```
#include <iostream>
```

- ```
using std::cin;
```

- ```
using std::cout;
```

- \* Can specify just the objects we want

- Used to resolve name clashes

- Programs use many classes, functions

- \* Commonly have same names

- \* Namespaces deal with this

- \* Can be "on" or "off"

- \*\* If names might conflict à turn off

# Example #1 : namespace

```
// namespaces
```

```
#include <iostream>
```

```
using namespace std;
```

```
namespace NS1
```

```
{
```

```
 int x = 5;
```

```
}
```

```
namespace NS2
```

```
{
```

```
 double x = 3.1416;
```

```
}
```

```
int main () {
```

```
 cout << NS1::x << endl;
```

```
 cout << NS2::x << endl;
```

```
 return 0;
```

```
}
```



5  
3.1416

# Example #2 : namespace

```
// namespaces
#include <iostream>
using namespace std;
namespace NS1
{
 int x = 5;
 int y = 10;
}

namespace NS2
{
 double x = 3.1416;
 double y = 2.7183;
}

int main () {
 using NS1::x;
 using NS2::y;
 cout << x << endl;
 cout << y << endl;
 cout << NS1::y << endl;
 cout << NS2::x << endl;
 return 0;
}
```



5  
2.7183  
10  
3.1416


# Example #3 : namespace

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

namespace NS1
{
 int x = 5;
 int y = 10;
}

namespace NS2
{
 double x = 3.1416;
 double y = 2.7183;
}

int main () {
 using namespace NS1::x;
 cout << x << endl;
 cout << y << endl;
 cout << NS2::x << endl;
 cout << NS2::y << endl;
 return 0;
}
```



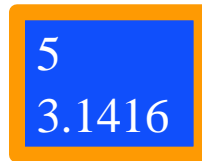
```
5
10
3.1416
2.7183
```

# Example #4 : namespace

```
// Using namespace
#include <iostream>
using namespace std;
namespace NS1
{
 int x = 5;
}

namespace NS2
{
 double x = 3.1416;
}

int main () {
 {
 using namespace NS1;
 cout << x << endl;
 }
 {
 using namespace NS2;
 cout << x << endl;
 }
 return 0;
}
```



5  
3.1416

# Hello World

```
/* Hello World in Java */
public class HelloWorld{

 static public void main(String args[]) {
 System.out.println("Hello World!");
 return;
 }
}
```

```
#include <iostream>

/* Hello World in C++ */
int main() {
 std::cout << "Hello World!" << std::endl;
 return 0;
}
```



# Standard Input and Output

- **Output stream cout**

```
std::cout << myVar;
```

- Object-oriented printing to console
- Built-in types can be printed using the left-shift operator
- Similar than System.out.print in Java but more flexible (stream modifiers; more later)

- **Input stream cin**

```
std::cin >> myVar;
```

- Object-oriented input from console
- Built-in types can be converted and assigned with the right-shift operator

# Using Definitions of the Standard Namespace

- **iostream library necessary for console input and output.**
- **Declarations are in the namespace std (standard).**

- Using a single declaration:

- just once

```
std::cout
```

- in the whole scope

```
using std::cout;
```

- Using all the declaration within a namespace in a scope (avoid!)

```
using namespace std;
```

# Main Function

- **C/C++ program entry point main which is of type**

```
int main(void);
int main(int argc, char *argv[]);
```

- **All source files in a project are allowed to define only one main function.**
  - Note: Visual Studio defines additionally program entry points (other “main” functions). Standard compliant C++ code will only use the above.

# Java and C++

- **Java**
  - Compiled to byte code
  - Executed by virtual machine
    - Object-oriented
    - Platform-independent byte code
- **C++**
  - Preprocessor
  - Compiled to object code
  - Linked to binary executable
    - Object-oriented, generic and functional features
    - Object code and executable are platform-specific

# C++ Fundamentals

- **Fundamental and complex data types including classes and strings**
- **Operators for fundamental types**
- **Control and decision statements**

# Variable and Function Names

```
identifier :
 underscore
 letter
 identifier following-character
```

```
following-character :
 letter
 underscore
 digit
```

```
letter : one of
 A B ... Z a b ... z
```

```
digit : one of
 0 1 2 ... 9
```

```
underscore : _
```

**Exactly like in Java**

**Case sensitive!**

**Examples:**

**i5**

**\_\_do\_not\_use\_\_**

**butUseThis**

**myFavoriteVariable**

# Declarations

- **Declarations introduce names into a program. Declarations may occur in different places in a program.**
- **What to declare?**
  - variables
  - functions
  - classes, structures and union components
  - types
  - type tags
  - enumeration constants
  - namespace
  - statement labels
  - preprocessor macros

# Definition vs. Declaration

- **Java and C++ provide definitions in one file and use it in many files**
- **Java**
  - Name is imported into another file.
- **C++ (Each file is compiled separately – if not #include'd)**
  - Linker ensures that name (according to scoping rules) refers to the same entity everywhere.
    - Definition allocates a variable.
    - Declaration introduces only the name.



# Fundamental Data Types

- **Three categories integral, floating and void.**
- **integral**
  - `bool, char, short, int, long, long long` (in C++11)
    - `intN_t` with  $N = 8, 16, 32$  or  $64$  (only C99);
    - MSVC: `_intN` with  $N = 8, 16, 32$  or  $64$
- **floating**
  - `float, double, long double`
- **void**

**... close to Java**

**BUT size may vary with C++ compiler/OS**

**Standard defines minimum sizes**

# Type Modifiers and Size

- **Modifiers**

- unsigned, signed, short, long

- **Sizes in MSVC++**

- 1 byte

- bool, char, unsigned char, signed char

- 2 bytes

- short, unsigned short

- 4 bytes

- int, unsigned int, long, unsigned long, float

- 8 bytes

- double, long long

- 18 bytes

- long double

# Derived Data Types

- **Directly derived data types**

- *Arrays, functions, pointers, object references, constants*

- **Composed derivative types**

***To be defined later !***

- *classes, structures, unions, scoped enumerations*

```
class myClass
{
...
};
```

# Automatic Typing with `auto`

- **Most often initialization can be done better (less error prone) by using `auto` types.**

```
auto iVal=65;
auto oiVal=iVal;
auto fVal=3.0f;
auto ofVal=fVal;
```

- **Aside: Arithmetic literals**

```
1 is an int
1U is an unsigned int
1L is a long
1LL is a long long
1.0f is a float
1.0 is a double
'\1' is a char.
```

# Compilers and IDEs

- [Apple Xcode C++](#)
- [Bloodshed Dev-C++](#)
- [Code::Blocks](#)
- [Cygwin](#)
- [Eclipse for C++](#)
- [MINGW - "Minimalist GNU for Windows"](#)
- [GNU CC](#)
- [The LLVM Compiler Infrastructure](#)
- [Microsoft Visual C++ 2010](#)
- [Sun Studio NetBeans](#)

# Libraries

- **General**
  - [Boost](#)
  - [MFC: Microsoft Foundation classes](#)
  - [STL: Standard Template Library](#)
- **GUI**
  - [MFC GUI](#)
  - [Qt](#)
  - [SFML](#)
  - [WxWidgets](#)

# Next week:

Java in C++

- **Basic Object-oriented C++**
  - Strongly-typed Enumerations
  - Operators, Ch. 4.1-4.9
  - Selection and Iteration Statements, Ch. 1.4, 5.3-5.5
  - Static casts, Ch. 4.11.3-5.12.6
  - Overview of `std::string`
  - Introduction to `std::array` and `std::vector`