Overview

1. History of C++
2. C++ characteristics
3. C++ vs. Java
4. Hello World
5. Variables and Data Types
6. Operators
7. Basic Input and Output
8. Control structures (if, switch, loops)
9. Functions

C++ History

- 1979: Bjarne Stroustrup began "C with Classes"
- 1983: Renamed to C++
- 1998: C++ became ISO standard (ISO/IEC C++98)
- 2003: ISO/IEC C++03
- 2011: ISO/IEC C++11

C++ Applications

- Game Engines
  Source Engine (HL2, CS:S), id Tech 4 and 5 (Rage, Doom 4)
- Mozilla
  Firefox, Thunderbird
- Google back-end
- Adobe Systems
  Photoshop, Illustrator
- Parts of Mac OS X, Facebook

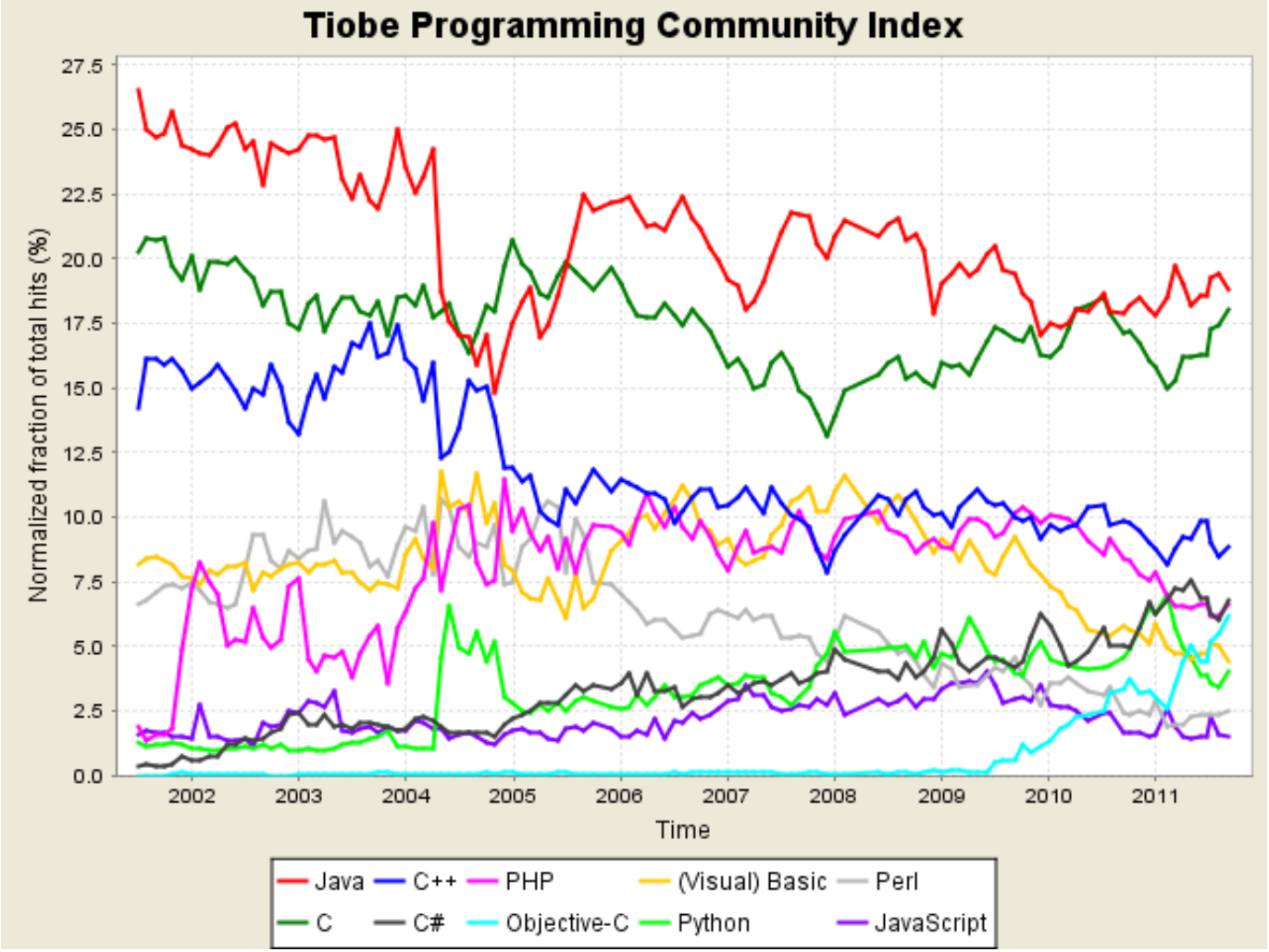
Bjarne Stroustrup

Why I created C++

C++ Characteristics?

- It's fast
  C++ is a superset of C, adds features like object-orientation, exception handling, generic programming, standard library
- Object-oriented programming
  Encapsulation, Inheritance and polymorphism, build cleaner and more concise code than you would in C
- Widely used

Top 10 popular programming languages (09/11)



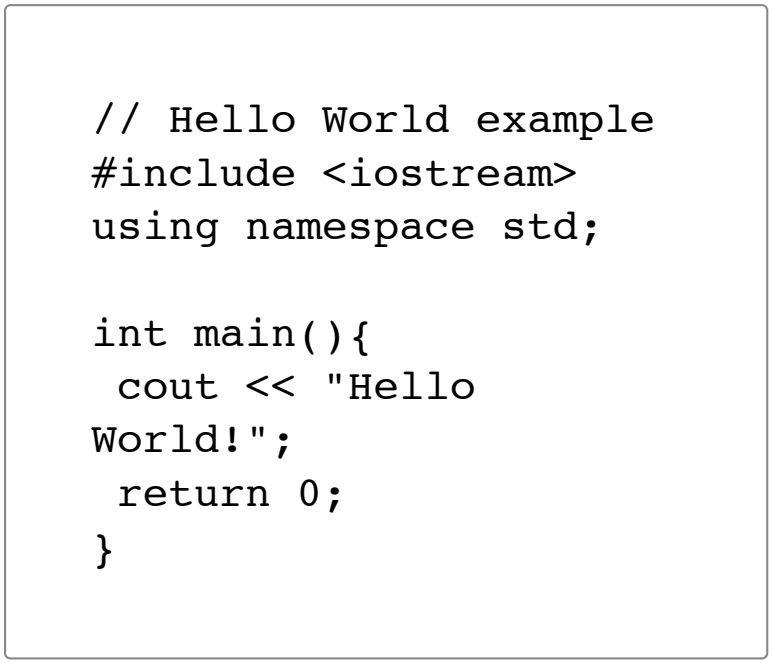
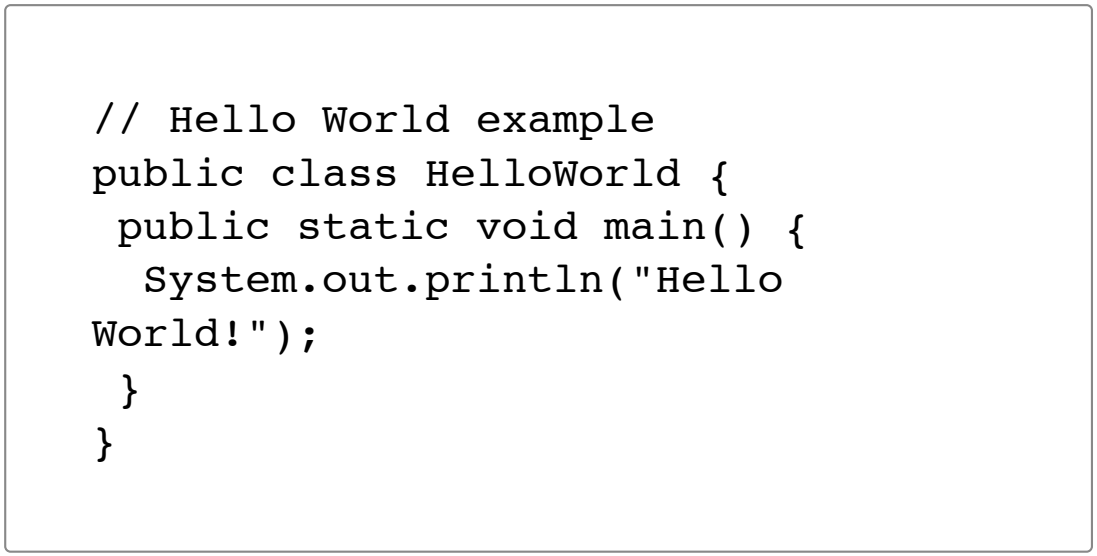
Rating based on the number of skilled engineers world-wide, courses and third party vendors. Used to make a strategic decision about what programming language should be adopted when starting to build a new software system.

C++ vs. Java

	C++	Java
Object-Orientation	YES	YES
Class-Orientation	YES	YES
Class Inheritance	Multiple Inheritance	Single Inheritance
Compiled to	Machine Code	Byte Code (JVM)
Garbage Collection	NO	YES
Runtime Checks	NO	YES

Garbage Collection http://stackoverflow.com/questions/147130/why-doesnt-c-have-a-garbage-collector

C++ vs. Java - Hello World

C++	Java
	

```
// Hello World example
#include <iostream>
using namespace std;

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

```
// Hello World example
public class HelloWorld {
 public static void main() {
  System.out.println("Hello
World!");
 }
}
```

Step-by-Step

a(href="http://www.cplusplus.com/doc/tutorial/program_structure/") Overview

```
// Hello World example
```

Comments

```
// This is a single-line comment

/* This is a single-line comment too! */

/*
 This is a multi-line comment
*/
```

Must not be nested

```
/*
 Nested comment /* are not allowed! */
```

```
 */
```

Step-by-Step

```
#include <iostream>
```

Preprocessor directive `#include`

```
#include <string> // Includes the string standard library
#include "Foo.h" // Includes a file named Foo.h
```

Step-by-Step

```
using namespace std;
```

Using a namespaces, see `cout`

Step-by-Step

```
int main() {}
```

Main function declaration, main point where C++ programs starts their execution

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.

Step-by-Step

```
cout << "Hello World!";
```

inserts a sequence of characters into the standard output stream (corresponds to the screen), elements in the C++ standard library are in the namespace `std`

With namespaces

```
using namespace std;
cout << "Hello World";
```

Without namespaces

```
std::cout << "Hello World";
```

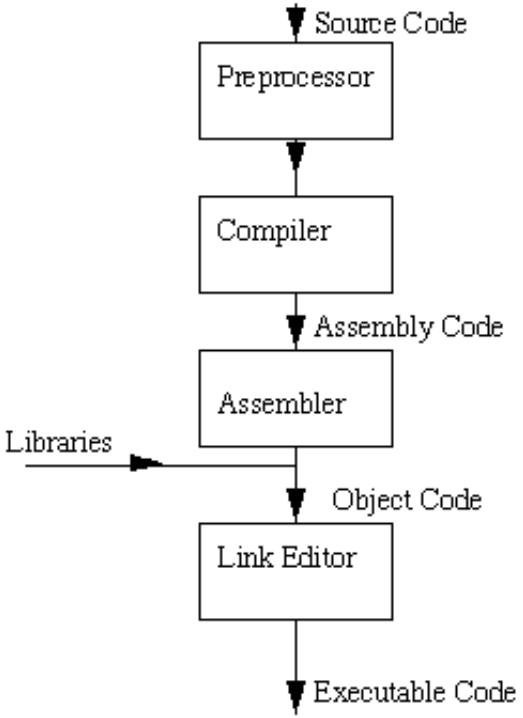
Step-by-Step

```
return 0;
```

Causes the main function to finish with return code 0, return code 0 is interpreted as the program worked as expected without any errors

C Compilation Model

- Pre-Processor
  Removes Comments, Interprets preprocessor directives #
- Compiler
  Translates source into assembly code
- Assembler
  Creates object code
- Linker/Link Editor
  Includes linked libraries, creates executable



Variables

```
float speed;
int _count = 0;
```

`datatype identifier = initial_value;`

- datatype
  e.g. fundamental data type: int, float, char
- identifier
  Sequence of one or more letters, digits or underscore characters. Have to begin with a letter or underscore character. Must not equal reserved keywords
- initial_value ALWAYS INITIALIZE A VARIABLE!
  Initialization is optional, but then value of a variable is undetermined

Fundamental Data Types (32-bit)

Name	Size
char	1 byte
short int	2 byte
int	4 byte
long int	4 byte
bool	1 byte
float	4 byte
double	8 byte
long double	8 byte
wchar_t	2 or 4 byte

a(href="http://www.cplusplus.com/doc/tutorial/variables/") Overview

Reserved Keywords

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while

Arrays

```
int quarterlySales[3];
int yearlySales[3] = {100, 200, 300};
```

```
type identifier[size];
```

Initialization

- Local arrays are not initialized to any value by default
- Global and static arrays are initialized with zeros

Accessing Values

```
int firstQuartalSale = quarterlySales[0];
```

No validity check of indices at run-time!

Strings

Using string class from C++ language library

```
#include <string>
...
string aString = "This is a string.";
cout << aString;
```

Special string characters

\n	new line
\t	tab
\\	backslash (\)
\"	double quote (")

Constants

```
#define PI 3.14159f // preprocessor directive

const int width = 10; // use of const keyword
width = 1; // results in a compiler error
```

textural replacement of defines

```
#define PRICE 3+4

float cost = PRICE * 4; // 3+4*4 = 19
```

Operators

Operator	Example
=	`a = 5;` `a = b = c = 5;`
+, -, *, /, %	`a+b` `a*b`

&, |, ^, ~, <<, >> Bitwise	`int a = 1, b =2; // 1 = 01, 2 = 10`
	`a & b  // 00`
	`a | b  // 11`
	`a ^ b  // 11`
	`~a  // 10`
	`a << 1  // 10`

```
b >> 1   // 01
```

Operators

Operator	Example

++, --	`int a = 0;`
	`a++ // 1`
	`a-- // -1`

==, !=, >, <, >=, <=, !, &&, ||	`(3 != 2)  // true`
	`(6 >= 6)  // true`
	`(5 < 5)  // false`

+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |= Compound assignment	`a += b // a = a+b` `a |= b // a = a | b`

http://www.cplusplus.com/doc/tutorial/operators/

In- and Output

Use streams to perform in- and output declared in library `iostream`

- Insert or extract characters to/from it
- Default output is screen
- Default input is keyboard

Output

Output with the insertion operator << and `cout`

```
#include <iostream>
using namespace std;
...
cout << "Hello World!";
cout << 10;
cout << a;
```

Insertion operator can be used more than once

```
cout << "Hello! My name is " << name;
```

No implicit line break, use `\n` or `endl` instead

```
cout << "Hello World!\n";
cout << "Hello World!" << endl;
```

Input

Input with the extraction operator >> and `cin`

```
#include <iostream>
using namespace std;
...
int a, b, c = 0;
cin >> a;

cin >> b >> c;
cin >> b;
cin >> c;
```

- Waits until the RETURN key is pressed
- Stops reading if any blank space is found

```
string line;
getline(cin, line);
```

String Conversion

Use library `sstream`

Convert string to integer

```
#include <sstream>
using namespace std;
int value = 0;
string myString = 10;

stringstream sstream(myString);
sstream >> value;
```

Convert integer to string

```
int value = 1;
string myString;

stringstream sstream;
sstream << value;
sstream >> myString;
```

Control Structures

- if/else
- loops (while, for)
- jump statements (switch, break, continue)

If/Else

```
if (condition) statement
```

```
if( a > 5){
 a = 10;
}

if( a < 5 ){
 a = 0;
}else{
 a = 10;
}

//shorter
a = a < 5 ? 0 : 10;
```

Use brackets to avoid mistakes!

Loops (for)

```
for (initialization; condition; increase) statement
```

```
for( int i=0; i<10; i++){
 cout << i;
}
```

More than one expression

```
int i,j = 0;
for ( i=0, j=100 ; i!=j ; i++, j-- ){
 cout << i << "!=" << j;
}
```

Loops (while)

```
while (condition) statement
```

```
int a = 5;
while( a < 5 ){
 cout << a;
 a++;
}
//Output:
```

```
int a = 5;
do{
 cout << a;
 a++;
}while( a < 5 );
// Output: 5
```

Jump statements (break and continue)

- End a loop even the conditions is true with `break;`
- Skip code with `continue;`

```
int i = 10;
while(i>0){
 i--;
 if(i % 2 == 0){ continue; }
 if(i == 5){ break; }

 cout << i;
}
// Output: 97
```

Switch

```
int a = 1;

switch (a) {
 case 1:
  cout << "a is 1";
  break;
 default:
  cout << "no case found";

}
```

First match of expression with constant is used and executed until the `break;` is found

Functions

`type name ([parameter,...]){ statements }`

Functions with no return value uses `void`

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

int add(int a,int b){
 return a + b;
}

void printValue(int value){
 cout << value;
}

int main(){
 int a = 4;
 int b = 5;
 cout << a << " + " << b << " = " << add(a,b);
 return 0;
}
// Output: 4 + 5 = 9
```

Functions

Why do we need functions?

- Cleanup code
- Make code reusable
- Hide complexity

Possible problems when

- modify arguments
- return multiple values

Functions

Function to encapsulate reading multiple values

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

void readCallByValue(int price, int amount){
 cout << "Enter a price: "; cin >> price;
 cout << "Enter the amount: "; cin >> amount;
}

void readCallByReference(int& price, int& amount){
 cout << "Enter a price: "; cin >> price;
 cout << "Enter the amount: "; cin >> amount;
}

int main(){
 int price = 0;
 int amount = 0;
 readCallByValue(price, amount); // price and amount is 0
 readCallByReference(price, amount);
}
```

Functions

Call by Value

- The values of the variables are copied
- Changes are made to the copies

```
void readCallByValue(int price, int amount){ ... }
```

Call by Reference (ampersand sign `&`)

- The references to the variables are passed
- Changes are made to the variables

```
void readCallByReference(int& price, int& amount){ ... }
```

Functions

- Functions can have the same name
- Compiler differentiates based on the number and type of arguments

```
int add(int a, int b){ ... }
int add(float a, float b){ ... }
```

- Default value of arguments

```
int sum(int a, int b = 0){ return a+b; }

int main(){
 sum(10);        // 10
 sum(10,2);      // 12
}
```