C++ vs. C

CSC100 / Introduction to programming in C/C++

Marcus Birkenkrahe

May 5, 2024

README

- This script introduces the basics of C++ and compares C and C++.
- PDF version of this file available in GitHub.
- This section, including some sample code, is based on Stroustrup (2014) and Hansen (2013).

History

- Both modern C and C++ descend from 1979's "classic" C.
- C++ is a superset of C: constructs that are both C and C++ have the same meaning (= semantics) in both languages.
- Notable exception: *character literal* (the byte size of a character constant) in C and C++

```
int s = sizeof('a');
printf("%d\n", s);
```

```
4
```

```
using namespace std;
int s = sizeof('a');
cout << s;
```

```
1
```

- C++ employs stricter *data type* checking: the language doesn't quite let you get away with as much.

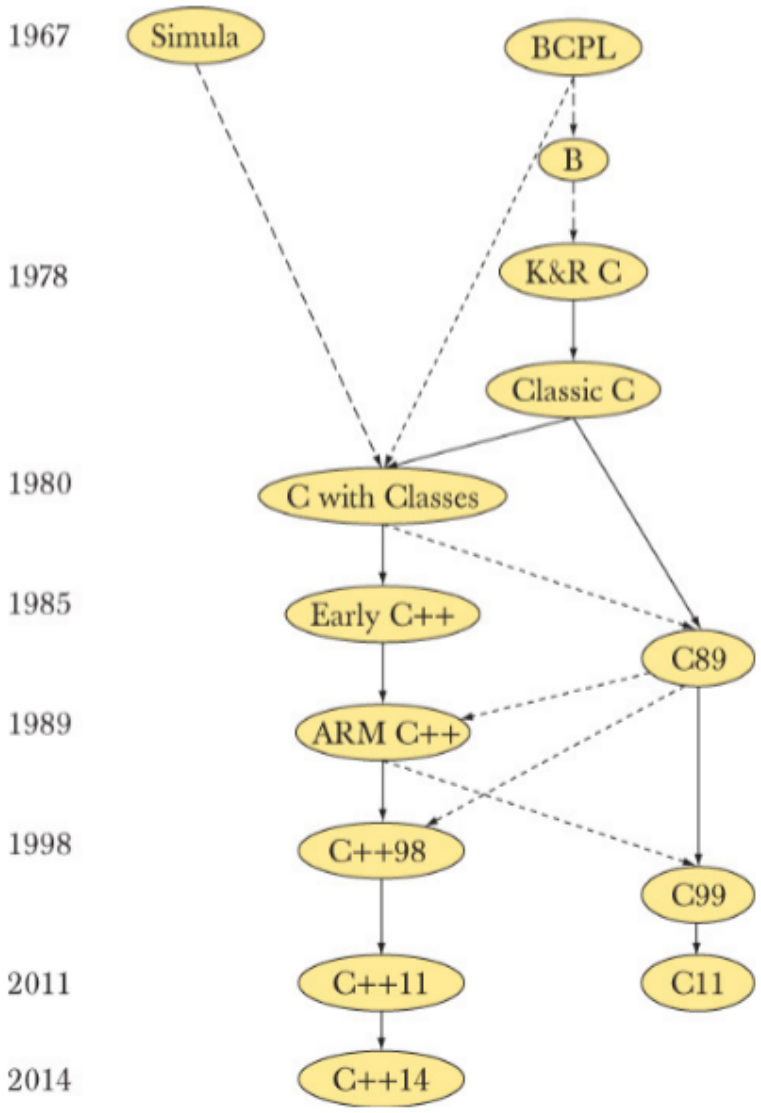

Figure 1: History of C++ 1967-2014 (Source: Stroustrup, 2014)

Sample program

- It all begins with `Hello World`, of course. When you tangle this file as `hello.cpp` you need to use `g++` to compile it (part of GCC).

```
#include <iostream>

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

```
Hello World!
```

- Org-mode requires the `C++` header option and `.cpp` file extension, otherwise everything is as usual:

```
std::cout << "Hello World!\n";
```

- Similarities and the differences between C and C++[1]:

C	C++	Difference
`stdio.h`	`iostream`	Header file
	`std_lib_facilities.h`	
`printf("...\n);`	`cout << "...\n";`	

- Compilation: the tangled code can be compiled and run using `g++`

```
$ g++ -o hello.exe hello.cpp
$ hello
```

- `g++` is part of `gcc` which we've been using all along:

[1] Stroustrup (2014) recommends `std_lib_facilities.h` instead. You have to download this file from his site. The hello world program now runs without having to specify where the `cout` function comes from. Yet another variation declares `std` as a `namespace` which means we don't have to explicitly declare it with every use of its functions.

```
~/Documents/GitHub $ which g++
c:/Program Files (x86)/mingw-w64/.../bin/g++.exe

~/Documents/GitHub $ which gcc
c:/Program Files (x86)/mingw-w64/.../bin/gcc.exe
```

- You can also see that our work with Emacs and Org-mode carries over 100% to C++. No need to bother with complex development environments like VS Code (Microsoft) or IDEs like CodeBlocks - IMHO you'll NEVER get the time you invest in these back, and the technology changes all the time without you being able to do anything about or with it.

Sample program: rectangle

- Here is the rectangle program in C++ - we'll look at the details below.

```
// ***************************************************
// Compute and print perimeter and area of a rectangle
// Input: length and width of a rectangle
// Output: perimeter and area of the rectangle
// ***************************************************

// Include input/output library
#include <iostream>

// Use standard namespace
using namespace std;

// Begin main program
int main()
{
  // Declare variables
  double length, width, area, perimeter;
  // Print program information
  cout << "Program to compute and print perimeter and "
       << "area of a rectangle." << endl << endl;
  // Set variable values for length and width
  length = 6.0;
  width = 4.0;
```

```
  // Compute perimeter = 2 * (length + width)
  perimeter = 2 * (length + width);
  // Compute area = length * width
  area = length * width;
  // Print input and output values
  cout << "Length = " << length << endl;
  cout << "Width = " << width << endl;
  cout << "Perimeter = " << perimeter << endl;
  cout << "Area = " << area << endl;

  return 0;
} // End main program
```

C++ features that are missing from C

C++ features	What to do in C
Classes	use `struct` data structure instead
Exceptions	use error codes, return values
Function overloading	give each function a distinctive name
References	use pointers
`bool` data type	use `int`
namespace	manage scope of functions, variables, types (useful for larger projects w/multiple libs)x

Variables

- Types of variables / data types

`int`	Short for integer; stores whole numbers
`char`	Short for character; stores a single letter, digit, or symbol
`bool`	Short for Boolean; stores true or false
`float`	Short for floating point number; stores numbers with fractional parts
`double`	Short for double precision floating point number; stores bigger numbers with bigger fraction

- Declaring and initializing variables

```
using namespace std;

int myVariable = 1;
double a = 2.2;
```

Constants

- Declaring a constant as a *literal* (non-variable)

```
using namespace std;

const float pi = 3.14;  // pi is the constant, 3.14 is the literal
float radius = 5, area;

area = radius * radius * pi;
cout << area;

78.5
```

- The good old `printf` works, too, of course:

```
const float pi = 3.14;  // pi is the constant, 3.14 is the literal
float radius = 5, area;

area = radius * radius * pi;

printf("%g\n",area);

78.5
```

Assignments

- What do C and C++ do when we try to add an integer to a string?

```
using namespace std;

int myValue = 4;
int yourVal;
string myString = "word";

yourVal = myValue + myString;
```

Error output:

```
error: no match for 'operator+'
(operand types are 'int' and 'std::__cxx11::string'
yourVal = myValue + myString
     ~~~~~~~~^~~~~~~~~~
```

And in C: no error - just a warning, and an output is generated:

```
int foo = 4;
int bar;
char myString = "word";

bar = foo + myString;
printf("%d\n", bar);
printf("%d\n", myString);

8
4
```

Output

- Output in C++ is done with the object `cout` ("console output"), which prints information to the screen.
- `<<` is the *insertion operator*
- `endl` (end line) is the equivalent of `"\n"`

```
using namespace std;

int myVariable = 1;
double a = 2.2;

cout << myVariable << endl;
cout << a;

1
2.2
```

- You can pipeline console output:

```
using namespace std; int myVal = 1000;

cout << "Go Scots! " << "You can do it!" << endl << myVal;

Go Scots! You can do it!
1000
```

  - You can still use \n:

```
using namespace std; int myVal = 1000;

cout << "Go Scots!\nYou can do it!" << endl << myVal;

Go Scots!
You can do it!
1000
```

- This makes formatting printout quite easy:

```
using namespace std;
int myVal = 1000;

cout << "Lyon" << endl;
cout.width(16);
cout << "College" << endl;
cout << "****************" << endl;
cout << "Freshmen/juniors" << endl;

Lyon
         College
****************
Freshmen/juniors
```

Input

- Generating an input file

```
echo "1000" > ../data/input
cat ../data/input
```

- To generate input, use the `cin` (pronounced 'see-in', "console input") object with the extraction operator `>>`.

```
using namespace std;

int x = 0;
cout << "Please enter a value for x " << endl;

cin >> x;   // this is equivalent scanf("%d", &x);

cout << "You entered: " << x << endl;

Please enter a value for x
You entered: 1000
```

- "Exception handling": Checking failed input with `cin.fail`. This time, no input was provided.

```
using namespace std;

int x = 0;

cout << "Please enter a value for x " << endl;

cin >> x;
if (cin.fail())
  {
    cout << "That is not a valid input" << endl;
  }

Please enter a value for x
That is not a valid input
```

Other differences:

There are slight differences in all areas we've covered:

Area	C++
Arithmetic	Similar basic arithmetic capabilities but supports operator overloading, allowing custom behavior for arithmetic operations on user-defined types.
Comments	Same as C, no additional comment features distinct from C.
Selection	Uses the same if, else if, else, and switch constructs, but with C++17, if and switch can also include initializer statements to declare variables within the statement.
Strings	Supports both C-style strings and the std::string class, which offers many utilities for string manipulation.
Loops	In addition to C-style loops, supports range-based for loops (for(auto x : container)) for easier iteration over containers.
Arrays	Supports C-style arrays and also introduces std::array and std::vector for safer and more flexible arrays that can dynamically resize.
Functions	Supports all features of C, plus member functions, overloading, and templates for generic programming.
Pointers	Adds smart pointers (std::unique_ptr, std::shared_ptr) which manage memory automatically and provide more safety and convenience.