C++ vs. C and OOP

CSC100 / Introduction to programming in C/C++

README

- This script introduces the basics of C++ and compares C and C++.
- PDF version of this file and of the completed practice workbooks is available in GitHub in the directory `10_cpp`.
- This section, including some sample code, is based on Stroustrup (2014) and Hansen (2013).

Overview

History

- Historically, modern C and C++ are siblings: both descend from classic C.

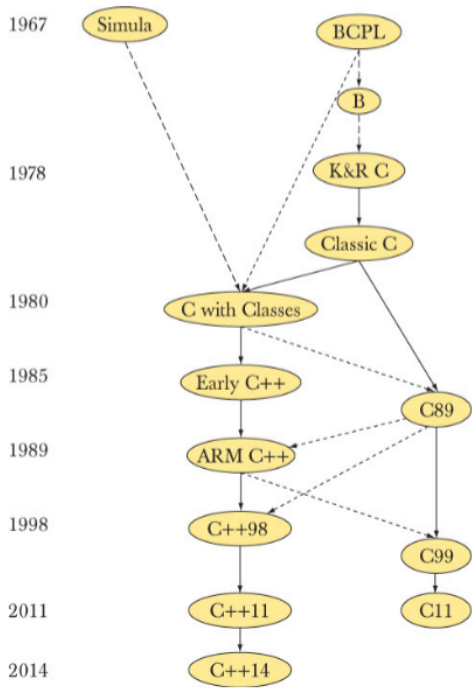


Figure 1: History of C++ 1967-2014 (Source: Stroustrup, 2014)

- 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
```

- `[ ]`

  What does this mean for the use of `sizeof` in `for` loops when looping over arrays?

  > You don't need to divide the `sizeof(a)` of an array a by the size of an array element, e.g. `sizeof(a[0])` to get a's length.

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

Sample program

- It all begins with `Hello World`, of course.

```
#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++:

C	C++	Difference
`stdio.h`	`iostream`	Header file
	`std_lib_facilities.h`[1]	
`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:

```
~/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 - you'll NEVER get the time you invest in these back!!!

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`

C++ Basics

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 fractions

- 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
```

Assignments

- When a variable is set with = the left side is the `lvalue`
- The thing on the right that's assigned is the `rvalue`

```
using namespace std;

int myVal, myVal1;

myVal = 0;        // assigning 0 to myVal
myVal1 = myVal;   // assigning myVal to myVal1
```

- Not allowed, because the `lvalue` does not refer to a place where we can store a value:

```
5 + 6 = myVal;  // illegal assigment
```

- 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!

```
int myValue = 4;
int yourVal;
char myString = "word";

yourVal = myValue + myString;
printf("%d\n", yourVal);
```

```
92
```

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
```

- Pipelining 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
```

- Formatted print example.

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

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

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

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
```

- 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:

- Arithmetics
- Comments

- Selection
- Strings
- Loops
- Arrays
- Functions
- Pointers

Object Oriented Programming (OOP)

The Mythical Man-Month

- "As a project's complexity increases, the number of man-months to complete it goes up exponentially." (Brooks, 1975)

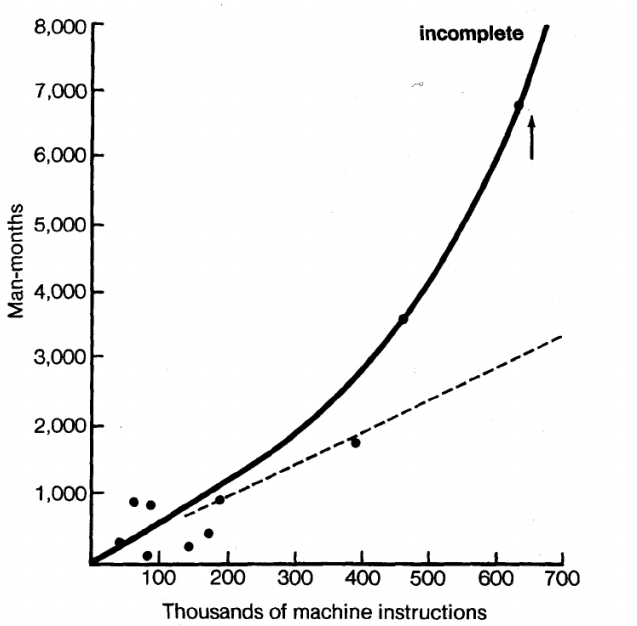


Figure 2: Source: The Mythical Man-Month, Brooks (1975)

- Software engineering struggles with the realities of software development, which is based on programming paradigms.
- [ ]

  What's a *paradigm*, especially in science?

  > A paradigm is a pattern or a model, a scientific paradigm is the set of concepts and practices that define a scientific discipline uses and is based on. According to Kuhn (1962), a paradigm shift leads to a scientific revolution when anomalies can no longer be explained using the old paradigm. Examples from physics: behavior of light as particle and wave (1900), Structure of the solar system (1500) (cp. Wikipedia).

Procedural programming

- **Procedural** programming is what you already know:
  - Programs are collection of *functions*
  - Data is *declared* separately
  - Data is passed as *arguments* to functions
  - Fairly easy to learn b/c of **modularization**
- Limitations of procedural programming:
  - Functions need to know the structure of the data
  - [X]

    Can you think of an example?

    ```
    int add (int x, int y)
    {
      return x + y; // this only works for int data
    }
    printf("%d\n", add(2,2));     // works well
    printf("%g\n", add(2.0,2.0)); // returns 0
    ```

    ```
    4
    8.48798e-314
    ```

  - Large programs become difficult to understand/debug

    - Large programs are hard to maintain/extend/reuse
- When an approach generates too many **anomalies**, a totally new approach, or a new **paradigm** often emerges - paradigms turn people's worldviews upside down.
- [ ]

  Can you think of *new paradigms* in science, history, etc.?

    - Darwin's model of evolution based on genetic mutations
    - Idea of climate change as man-made phenomenon linked to CO2
    - Focus on germs as the origin of disease
    - Cosmological model of the universe (and heliocentric model)
    - Relativity (special: of mass/energy, general: mass/spacetime)
    - Quantum mechanical model of the world at smallest distances

  Note: none of these are true in the "biblical" sense but they are scientifically true, that is they describe some of the world as an approximation, through abstraction, and are in continuous development.

Object orientation (SE concept)

- The greatest conceptual and practical difference between C and C++ is the explicit use of *object orientation* (OO).
- OO can extend to general design, analysis, testing, even management - whenever you focus not on the procedure but on the *objects* involved and their ability to exchange *messages*.
- *Classes* model real-world domain entities (modeling), e.g.
    - for a school application: `student`, `professor`, `course`, etc.
    - for a photo application: `slideshow`, `location`, `photo` etc.
- Higher level of *abstraction* during development (less detail)
    - When coding a `student` class, think about what a student, as an instance of the class, might do (*method*) or have (*attribute*)
    - You need to concern yourself with interactions and relationships between the different objects of your world
- [ ]

  What are examples for *methods* (= abilities) of a `student` class?

    E.g.

    - `student.enrol()`
    - `student.attend()`
    - `student.graduate()`
    - `student.dropClass()`
    - etc.

- [ ]

  What are examples for *attributes* (= properties) of a `student` class?

    E.g.

    - `student.name`
    - `student.level`
    - `student.grade`
    - `student.gender`
    - `student.enrolled`
    - etc.

- To compute things, e.g. find out if a student is registered this term, I can send a message to an *instance* of the `student` class, e.g. the student `Frank`, and ask him if he's registered this term:

```
Student Frank;   // Frank is a student
cout << Frank.enrolled();  // is Frank enrolled?
```

- This is very different from procedural programming where I would have to pass the student to that function:

```
int enrolled(student) {...} // function definition
int status = enrolled("Frank"); // check Frank's status
```

- The function depends on the business logic, as does the method of the Student class, but it is defined on *one* place - one change is enough.
- Objects contain data + their operations (= *encapsulation*)
- All of this is a little like developing your own video game (C++ based engines dominate video game and graphics development)[2]
- OOP is used successfully in very large program applications

OOP concepts (overview)

- Information-hiding via *encapsulation* (e.g. `student.enrolled()` hides specific implementation from users)
- *Inheritance* = creation of new classes (e.g. `IntStudent` as a class derived from `Student`.)
- *Polymorphism* = add new logic to a derived class without touching the original class (e.g. for `IntStudent.applyVisa()`).

Here is an example of how this looks like in UML (a modeling language, like BPMN):

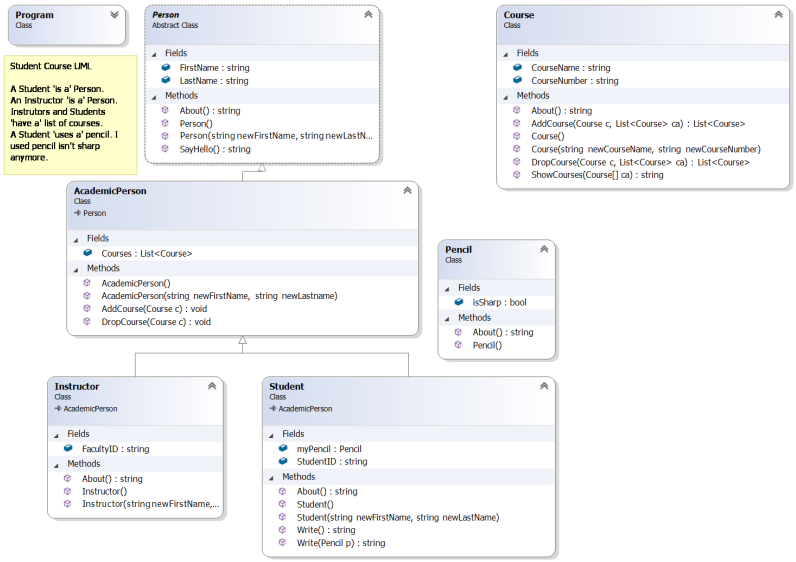
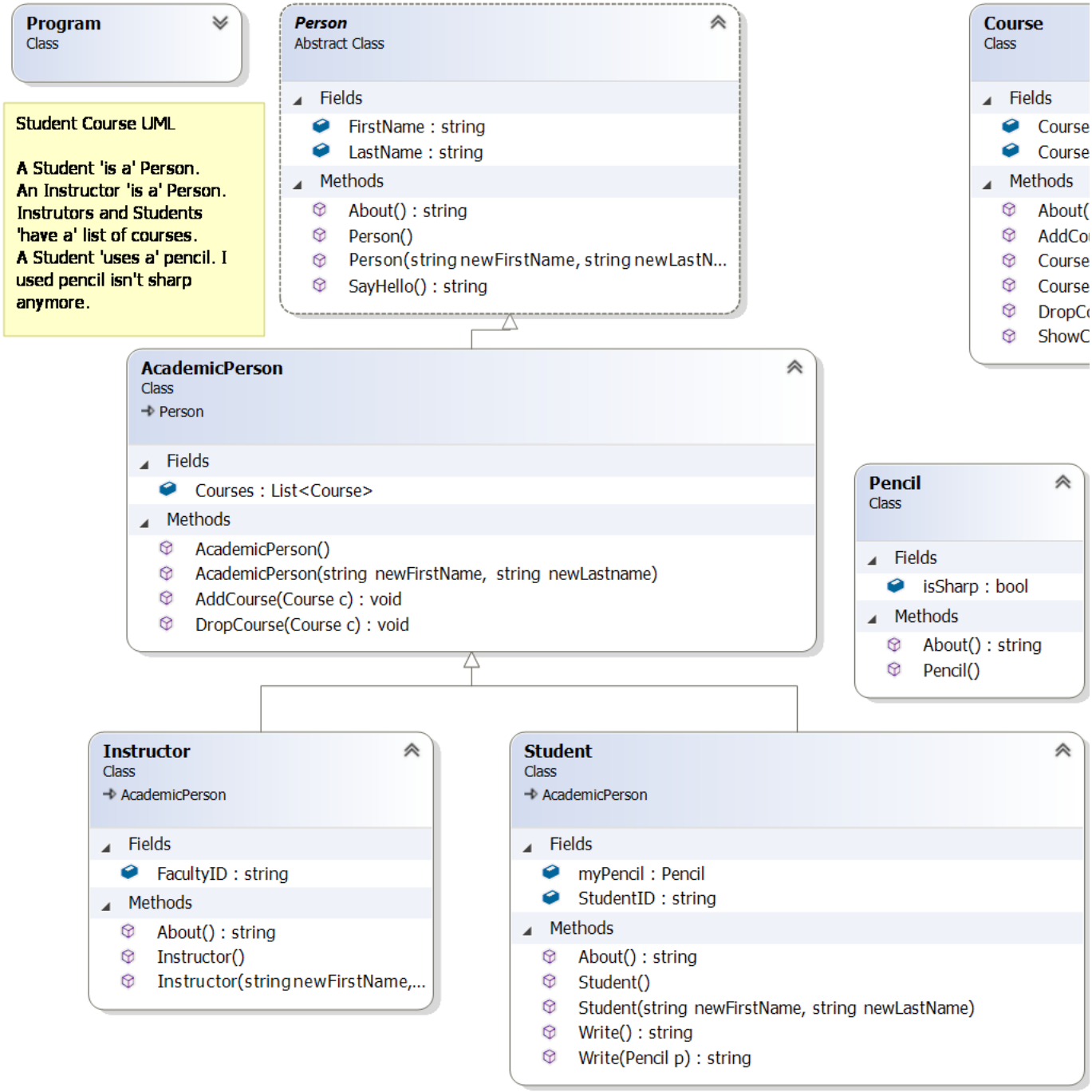


Figure 3: Class diagram (UML, source: Columbia U).

(Link:



)

Limitations

- OO Programming does not make bad code better
- Not everything decomposes into a class
- Steeper learning curve (especially for C++)
- Upfront investment because of design requirements
- Programs are larger, slower, more complex

Further study

- The "Rook's Guide to C++" (Hansen, 2013) which is freely (and legally - Creative Commons license) available as a PDF online covers the basics of C++ in 130 pages.
- Much more thorough is the book by Stroustrup (2014). It's expensive (though copies are floating around, and I got one copy for the library). It contains 1200 pages of C++ goodness.
- For a quick, high ROI overview of C++ in 40 min only, check out Mike Dane's "C++ Programming | In One Video" (2017). Annoying: ads. Talk about OOP begins about 30 min into the course. You may infer that about 1/3 of C++ is not C, which is about right.
- FreeCodeCamp offers a free C++ course on YouTube (2022), which leads to advanced topics - and takes 31 hours to watch. Uses VS Code editor with GCC and explains how to set it up.

- Udemy offers this 46-hour video-based course (2022) which is very nicely presented, contains exercises, but costs a little money (I got it for $10).
- See also "How to teach yourself programming in 10 years", or "Why is everyone in such a rush?" by Peter Norvig (director of research at Google and author of the standard textbook on AI, 2021).
- History and context: listen to the 2 hour podcast/interview with creator of C++ - Bjarne Stroustrup: C++ | Lex Fridman Podcast #48 (2020), which contains a wide range of C++ and programming related issues. (Lex Fridman is an AI/ML professor at MIT.)

References

- Brooks (1975). The Mythical Man-Month, Addison-Wesley. URL: fermatslibrary.com (extract)
- Hansen (2013). The Rook's Guide to C++. URL: rooksguide.org.
- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.
- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org
- Stroustrup (2014). Programming – Principles and Practice Using C++. Addison Wesley. URL: stroustrup.com.

Footnotes:

[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.

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

Yet another variation declares `std` as a `namespace` which means we don't have to explicitly declare it with every use of its functions:

```
using namespace std;
cout << "Hello World!\n";
```

[2] This is also why I got started in C++ rather than in C: for my PhD, I had to develop a large library of graphical objects (which in turn represented particle physics entities), and C++, which had only been developed a few years earlier, was just the right tool for that. Neither Java (1995) nor Python (2000) existed at the time!

Author: Marcus Birkenkrahe
Created: 2022-06-22 Wed 12:07