Programmazione

Prof. Marco Bertini
marco.bertini@unifi.it
http://www.micc.unifi.it/bertini/

Introduction

Why OO Development?

- Improved structure of software – easier to:
  - Understand
  - Maintain
  - Enhance
- Reusable software components & modules:
  - Enhance an existing component
  - Make new components more generalised

Structured vs. OO approach

- Structured analysis and design (or structured programming) mostly focussed on procedural structure and as such, functionality relating to the same data was still often spread throughout the software system.
- The object oriented paradigm focuses on structuring the system around the data by ***encapsulating*** data inside ***objects*** so that all functionality relating to that data is, in theory, contained within the object itself.

3 Key OO concepts

- Main idea: encapsulate data inside an “Object”

1. **Data Abstraction**: data is now seen in terms of the operations that we can perform on that data and not only in terms of the pure data values.

    - ADT’s, information hiding: because the details of the data formats and how those operations are actually implemented are hidden from other code that uses an object.

    - C++ classes: the principle facility used to implement data abstraction.

3 Key OO concepts

- Main idea: encapsulate data inside an “Object”

1. **Data Abstraction**: data is now seen in terms of the operations that we can perform on that data and not only in terms of the pure data values.

Abstraction is a good engineering tool: it allows to specify the external properties of an entity without having to specify the internal details. Thus, we can use the entity without having to deal with its implementation details.
Designing a complex system becomes easier, since we have to deal with its different levels of abstraction without having to deal with the more complex details of the levels below.

3 Key OO concepts - cont.

2. Inheritance (generalisation/specialization)

- **Class hierarchies**, **inheritance**: some classes of objects are generalizations of other more specific classes of objects; therefore it makes sense to *i)* put general functionality into what we refer to as a **base class** and *ii)* to allow more specific functionality to be placed in classes derived from the base class. The derived classes are said to inherit (or extend) functionality from the base class.

3 Key OO concepts - cont.

3. **Polymorphism**: we can use different types of objects in the same way and the software system – in this case C++, but could be Java or Python – will work out for us which functionality should actually be invoked, according to the actual type of object that is involved in a particular usage.

   - Same operations on different classes/objects: arithmetic operators that can apply to both integer and floating point numbers, although the actual implementation of arithmetic is quite different.
   - Virtual functions
   - Operator overloading

   In C++ polymorphism is supported on objects using what are called virtual functions as well as through further overloading of standard operators.

Objects and Classes

- An **object** is like an ordinary variable: is an actual instance of a software entity that holds real information, perhaps about something in the real world.
  - Holds information.
  - Software realization of some real world "thing".
  - Can have certain operations applied to it.

Objects and **Classes**

- A **class** is (like) a type:
  - Abstraction of a set of objects that behave identically: brings together the implementation of data and operations.
  - Defines internal implementation, operations.
  - Can create (**instantiate**) objects from a class: just like for a built-in type (e.g. `int` or `float`), a user defined type using the class facility can then be instantiated into objects or variables.

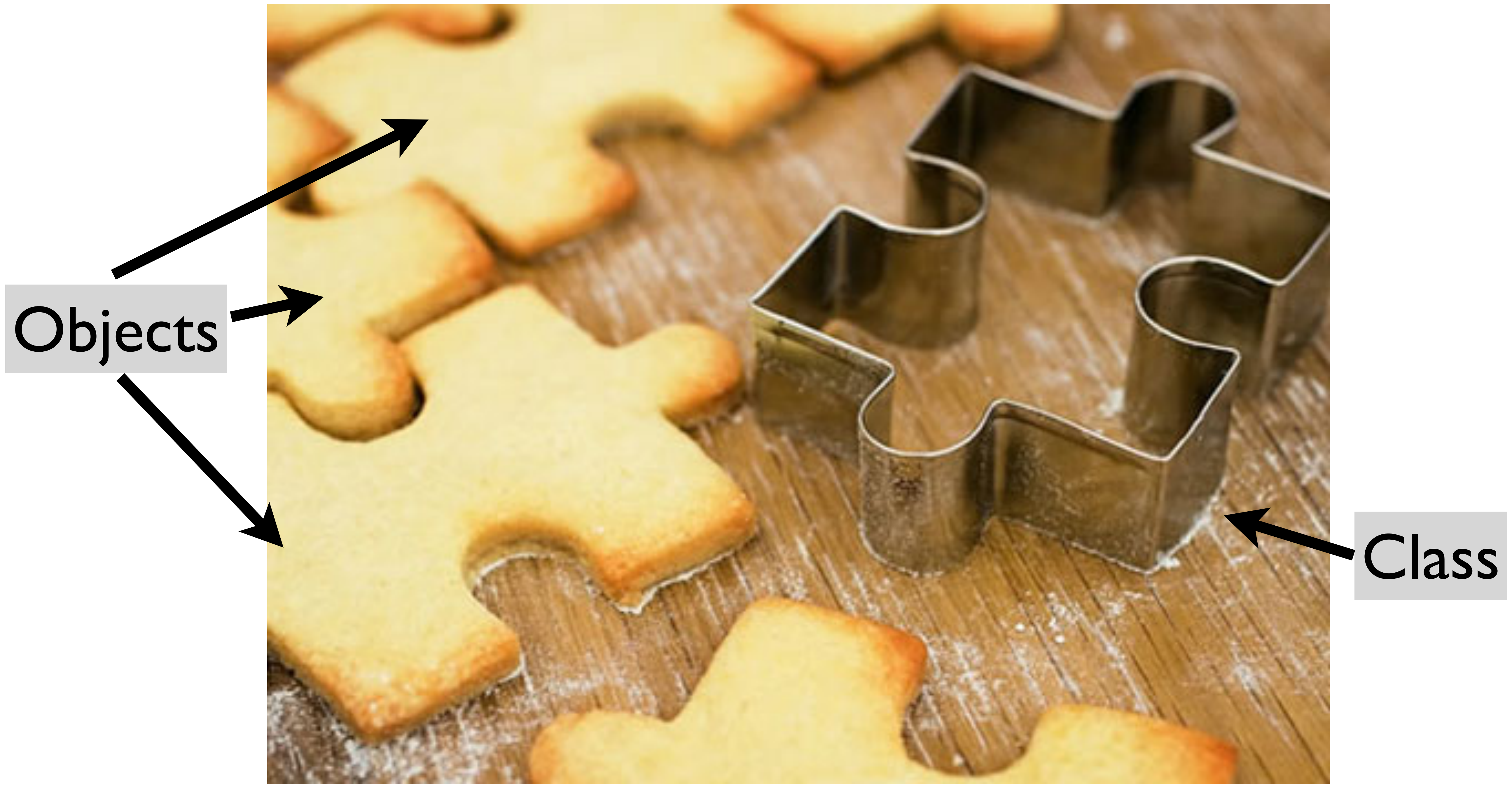
Objects and Classes

- Classes help you to organize your code and to reason about your programs.
- A **class** is the representation of an idea, a concept, in the code. An **object** of a class represents a particular example of the idea in the code.
- Without classes, a reader of the code would have to guess about the relationships among data items and functions - classes make such relationships explicit and "understood" by compilers. With classes, more of the high-level structure of your program is reflected in the code, not just in the comments.

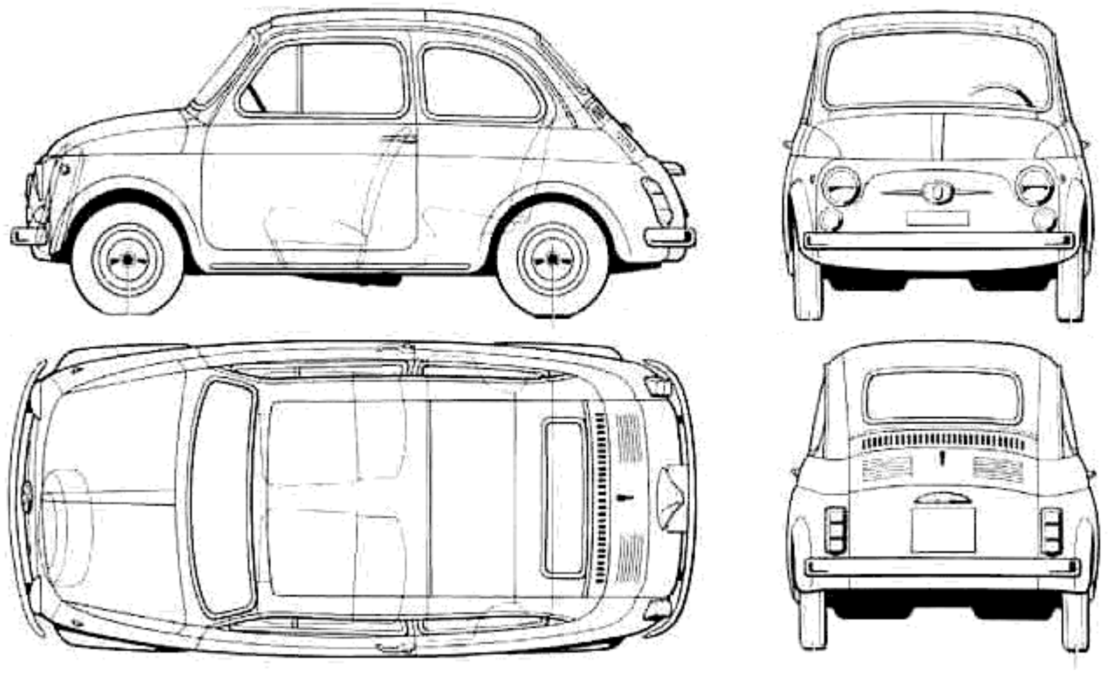
Objects and Classes

Objects and Classes

Objects and Classes



Objects and Classes



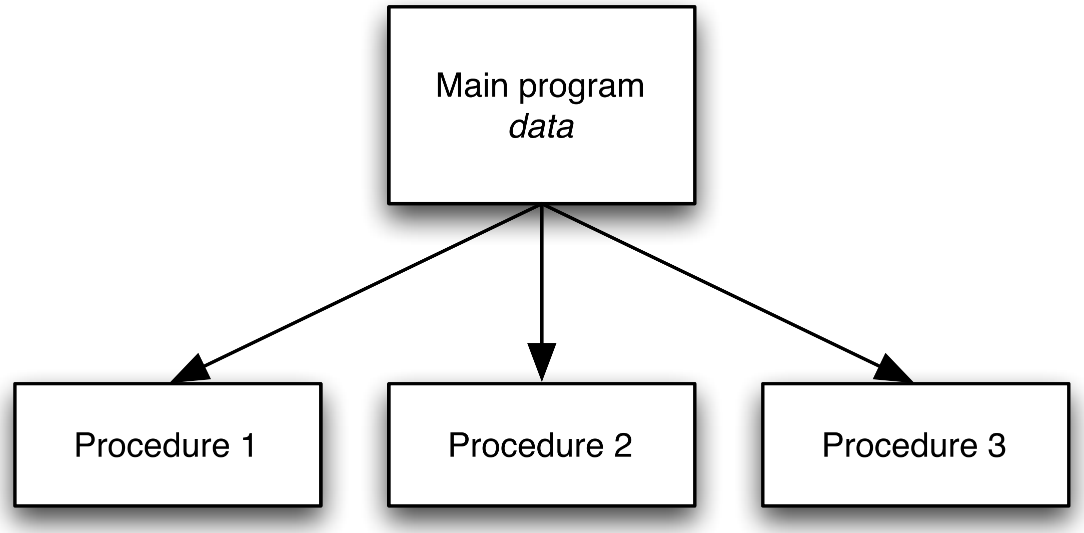
Class

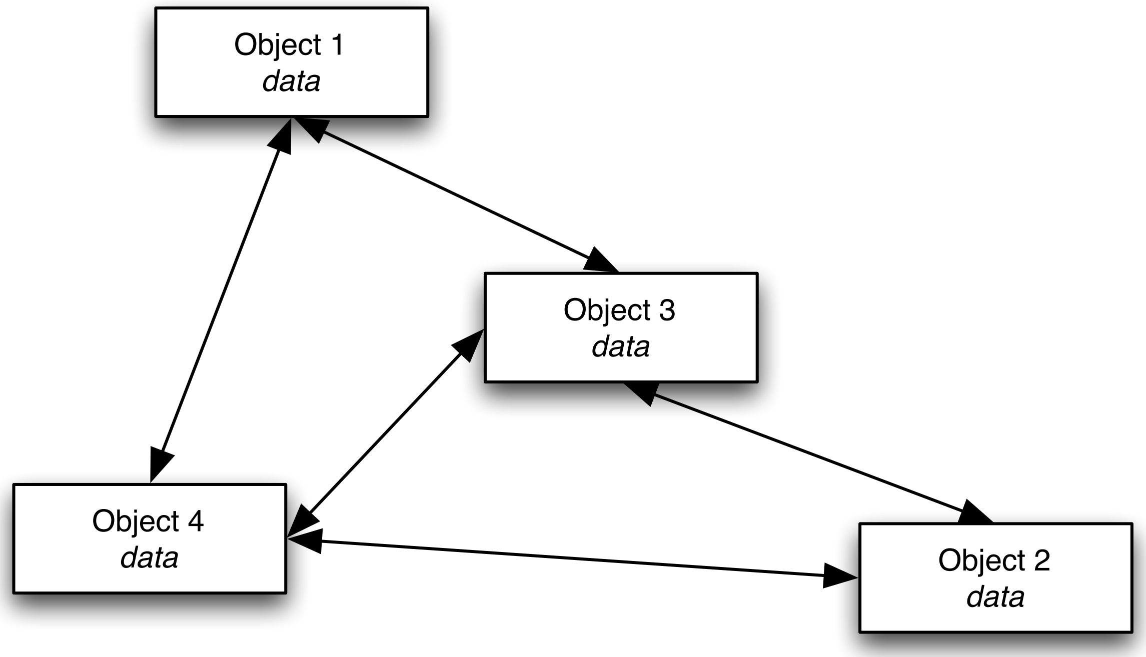
Object

Procedural Programming vs. OOP

- A program can be viewed as a sequence of procedure calls. The main program is responsible to pass data to the individual calls, the data is processed by the procedures.

- A program can be viewed as a web of interacting objects, each house-keeping its own state





Objects without Classes

- JavaScript is an object-oriented language that does NOT have classes. It uses prototypes (it’s called prototype-based or instance-based).
  - The inheritance of class-based OO languages becomes behaviour reuse, through “decoration” of existing objects which serve as prototypes.
  - ECMAScript 6 has introduced classes also in Javascript.

OO methods

- OO methods are a set of techniques for analyzing, decomposing and modularizing software systems architectures
- In general, systems evolve and functionality changes, but the objects (and classes of objects) tend to remain stable over time.
- The object oriented paradigm influences the entire software development process. It proceeds through a set of phases, although the boundaries are often blurred.

OO Software Development

- OO Analysis:
  - Identifying required functionality, classes and their relationships: often uses tools drawn from the Unified Modelling Language (UML) such as class diagrams and use cases.
- OO Design:
  - Specifying class hierarchies, class interfaces and class behaviour: UML tools such as class diagrams, interaction diagrams and state diagrams can be used in OO design.
- OO Programming:
  - Implementing an OO design in an OO programming language: in this phase code is actually written, tested and integrated with other code.

OOA

- Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system.
- OOA looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed and understanding the requirements.
- Analysis models do not consider any implementation constraints that might exist.

OOD

- Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications.
- During OOD developers create abstractions and mechanisms necessary to meet the systems' behavioral requirements determined during OOA
- The concepts in the analysis model are mapped onto implementation classes. The result is a model of the solution domain, a detailed description of how the system is to be built.

OOD - cont.

- OOD is relatively independent of the programming language used
- OOA focuses on what the system does, OOD on how the system does it.

Goals of design

- Create software that is easy to change and extend, i.e. flexible.
  - The goal is to attain highly cohesive, loosely coupled software.
- Decompose the system into modules, determining relations between them, e.g. identifying dependencies and form of communications
  - Determine the required classes, their use, their inheritance and composition relationships.

Object Oriented Programming

- Object-oriented programming (OOP) is primarily concerned with programming language and software implementation issues
- OOP is a programming paradigm that uses "objects" – data structures consisting of data and methods together with their interactions – to design applications and computer programs.
- An object-oriented program may be viewed as a collection of interacting objects, as opposed to the structural model, in which a program is seen as a list of tasks (subroutines) to perform.

C++

"I invented the term 'Object-Oriented', and I can tell you I did not have C++ in mind."

- Alan Kay

History of C++

- C++ invented by Bjarne Stroustrup, early 1980's
- C++ is not just an Object-Oriented Programming Language, it's multi-paradigm
- First commercial release 1985 was a C preprocessor.
- ISO Standard in 1998 (C++98), updated in 2003
- New standards: C++11, C++14 and C++17 (check your compiler)

C++ and C

- C++ is a direct descendant of C that retains almost all of C as a subset.
- C++ provides stronger type checking than C and directly supports a wider range of programming styles than C.

C++ sub-languages

- C++ can be considered as a federation of languages; the primary sublanguages are:
  - **C**: since C++ supports every programming technique supported by C (e.g. Procedural)
  - **Object Oriented C++**: with classes, encapsulation, inheritance, polymorphism, etc.
  - **Generic Programming C++**: C++ templates
  - **Functional**: C++11 has introduced lambdas

The first C++ program

Makefiles

- Larger (real world) projects will require to use a Makefile.
- A makefile is a text file (that is referenced by the `make` command) that describes the building of targets (executables and libraries), and contains information such as source-level dependencies and build-order dependencies.
  - Eclipse is an IDE that handles these makefiles

Creating Makefiles

- Makefiles can be created from other programs that tell how to create them.
  E.g. CMake is a program that creates *makefiles* that in turn tell how to create a program.
- CLion is an IDE that is based on CMake.

The C++ “Hello world”

```
#include <iostream>

int main() {
   // let’s greet
   std::cout << "Hello, ";
   std::string name = "world";
   std::cout << name << std::endl;
}
```

To compile:

```
g++ -Wall –o hello hello.cpp
```

Reading material

- L.J. Aguilar, “Fondamenti di programmazione in C++. Algoritmi, strutture dati e oggetti” - cap. 1
- D.S. Malik, "Programmazione in C++” - cap. 1
- Thinking in C++, 2nd ed. Volume 1, cap. 1 pp. 23-30

Credits

- These slides are (heavily) based on the material of Dr. Ian Richards, CSC2402, Univ. of Southern Queensland
- Dr. Douglas C. Schmidt, Washington University, St. Louis