1 Introduction¶

This module gives a broad overview of the main C++ features that differ from C, to allow you get started with C++ programming. We will briefly introduce, among certain other things, **reference variables**, **string** and **vector** data types, and **C++ namespaces, classes and objects**. In the later modules, we will thoroughly look into these concepts.

Contents

1.1 C++ Programming Language¶

C++ was first developed by **Bjarne Stroustrup** in the late 1970s, originally by name "*C with classes*". One of the main initial motivations for C++ was to introduce Object-oriented programming model to C. Eventually C++ has gathered a variety of other extended features, and is much more versatile language than C. Many people find these features useful, although some people also criticize C++ for its complexity.

> Point of interest iconObject-oriented programming
> previous | next
>
> Object-oriented programming was invented in the 1970s with the SIMULA language, initially designed for simulations and graphics programming. Unlike e.g. the *procedural programming* approaches, where programs are structured around *procedures* or *functions* that interact with each other to manipulate data, in *object-oriented* programming the software is designed around *objects* that consist of *attributes* (or data), and *methods* (or functions) that operate on object's attributes. The software logic is built of multiple objects that interact with one another. *Object-oriented approach is helpful in designing modular software through fairly independent objects.*
>
> There are many object-oriented languages, such as **Java** or **Smalltalk**, but in this course we will focus on **C++**. C++ is not a pure object-oriented language, and it also allows other programming paradigms, possible mixed with object-oriented programming. However, *C++ is popular on systems that require performance or better access to computer system resources.* **C++ is also backwards compatible with the C language, which allows easy integration with the large number of existing systems implemented in C.**
>
> In class-based object-oriented systems, the objects are defined by classes. A **class** defines a model, that can be instantiated as number of **objects**. Class defines the types of attributes that objects of that class will have, and the methods (or functions) that can be used to operate on the class. Typically, a class is designed to implement *information hiding*: an outsider should not be directly able to modify the state of an object that represent a class, but rather use the public function *interface* defined in class definition to interact with the objects. Information hiding is helpful in maintaining large software consisting of hundreds of classes and different systems. Implementation details of class internals can be modified without affecting the other parts of the system, as long as the interface stays the same.
>
> For example, we could have class for **Car**, that would have attributes for *speed*, *weight* or *passengers* (among many others). We may have multiple *objects* that represent class *Car*, such as "Thelma's Ford", "Nalle's Rolls", or "Elon's Tesla", each with unique values for *speed*, *weight* and *passengers*. The state of the objects can be altered with two methods: *accelerate* that affects the speed of the particular object, (e.g.: **Nalle's Rolls.accelerate()** would increase the speed of the car), and *drop_passenger* that will remove one passenger from the car. As a safety mechanism, the method implementation could also check that the car is stopped before dropping a passenger is allowed. You can see how different objects of the same class can have different attributes values in the following figure.
>
> ../_images/basics-classes.svg
> A large object-oriented software system could comprise of hundreds of different classes that interact with each other, and even larger number of objects representing these classes. Therefore careful design and awareness of proven design patterns are needed to keep the system maintainable.

C++ can be seen as an extension to C, although there are some details that may prevent a C program from directly compiling under C++ compiler, for example because C++ has extended set of keywords that could have been used as variable names in a C program (e.g., **new** and **delete**). The basic syntax, structures and data types from the C language are still available in C++.

C++ is standardised by the International Standard Organization (ISO). The C++11 standard brought in multiple new features to the language. The C++14 and 17 standards, that came after it, do not bring many changes. C++14 and 17 features can be easily found through a websearch. Most of the changes (if not all) can be found in this github repo and this compiler support listing. This material assumes **C++17** (ISO/IEC 14882:2017), the 2017 version of the standard, although new features added by C++14 and 17 standards to C++11 are not needed for this course.

See also

The wikipedia page on C++

1.1.1 C vs. C++¶

While C++ has its roots in the C language, it has grown the be quite much more extensive.

What is common?¶

- **Syntax**: statements end in semicolons; blocks are indicated as curly braces; everything is built inside functions (that in C++ may be inside classes)
- **Static data types**: Data types and their validity are checked at compile time. Therefore all variables, function arguments, etc. need to be specified with an explicit data type that is fixed at compile time. In C++, though, object inheritance and polymorphism add some flexibility to this.
- **Headers**: Data types, functions and other names shared between program modules are defined in headers, that are included in the beginning of program. Also standard library functions are in headers.
- **Compilation**: programs are first compiled into machine-executable code, linked into a single program, and executed separately.
- **main function**: execution starts from the *main()* function that must exist in all programs.
- **Direct memory access**: While C++ provides tools help avoiding memory access errors, it is still possible to have a C++ program access invalid memory, causing segmentation fault, or in worse case, invalid program behavior (and possible security flaws)

What is different?¶

C++ provides many additional features compared to C, such as:

- Object-oriented programming
- Generic programming with templates
- Extensive standard library with ready-made tools for different data structures, memory management, and so on.
- Tools for safer dynamic memory management
- Exceptions
- Nested namespaces

These extensions make it easier to manage large software, but also cause C++ to be rather complex programming language to master. For example the templates cause the compiler output messages to be sometimes very lengthy and difficult to understand.

1.2 About the course¶

In this course, the principles and concepts of the object oriented programming in C++ programming language are covered. As you will see, C++ is not the easiest language; it has a large set of features and C++ compilers occasionally produces long (and cryptic) compiler output messages. However, C++ is worth learning, because it is widely used: e.g. web browsers, many game engines, and graphics software, and is useful in situations where peformance matters. C++ programming is an often sought skill when recruiting programmers for projects in industry and academia.

Warning

This course is not designed as an elementary or first programming course. We assume that you are already familiar with basics of computer programming paradigm.

We also assume that you are familiar with C language, its syntax and basic concepts. We will not go through, for example, the basic C datatypes or pointer arithmetics on this course. In case you need to study more about C, the online course material is available along with many other information sources. The recommended main reference is "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie.

Two books, which are based on C++11, are worth mentioning as good sources of information:

- **C++ Primer** (5th edition, Addison-Wesley, 2012) by *Stanley N. Lippman, J. Lajoie and Barbara E. Moo* is an excellent and extensive course book. We refer to this book in several places of the material for additional information.
- **The C++ Programming Language** (4th edition, Addison-Wesley) by *Bjarne Stroustrup* is a complete reference by the author of C++. *This book is perhaps a bit less course-book-like, but it is a very solid package of information about all aspects of C++.*
- In addition, in several places we refer to the [cppreference] and [cplusplus] sites, which contain a reference to different C++ standard library classes and other information.

Typically, by convention, the C++ source files are named with "**.cpp**" file name suffix, to separate them from other languages, particularly C. Similarly, by convention, the C++ header files are named with "**.hpp**" suffix. These are just naming conventions, and the compiler would allow also other styles of naming. In different environments different naming styles may be used, but on this course we will follow this naming convention.

This material and the exercises roughly follow the Google C++ Style Guide. This is one of many different style guides for C++ and is in no way an industry standard (there is none for C++). It is, however, important to follow a specific style for each project.

cppreference
https://www.cppreference.com/

cplusplus
http://www.cplusplus.com/