In c: gcc -std=c89

In c++: g++ -std=c++98

E.g., g++ -std=c++98 -pedantic -Wall -o hello hello.cpp

Ending of code related to c++:

Implementation : .cpp, .cc, .cxx, .C

Header, hpp, hh, .hxx

Int main(void) in c → int main () in c++

Namespaces:

Naming collision (ODR- one definition rule)

In c, we didn't have namespace. Therefore, it was safe to type a long and safe name as struct EFSVorentwicklungsPunkt2D than a struct Punkt. But it c++ we have namespace.

How to open a namespace:

See these examples:

```
#include <stdio.h>

namespace schulung {

void foo() {
  puts("foo");
}

namespace deeper {

void foo() {
  schulung::foo();
  puts("bar");
}

} // Ende Namespace "deeper"

void baz() {
  foo(); // ::schulung::foo
  deeper::foo();
}

} // Ende Namespace "schulung"

int main() {
  schulung::baz();
  return 0;
}
```

If you divide it into header and implementation file, it will look like this:

```
/*** foobar.h ***/

#ifndef SCHULUNG_FOOBAR_H
#define SCHULUNG_FOOBAR_H

namespace schulung {

void foo();
void baz();

namespace deeper {

void foobar();

} // Ende Namespace "deeper"
} // Ende Namespace "schulung"

#endif
```

```
/*** foobar.cpp ***/

#include "foobar.h"
#include <stdio.h>

void schulung::foo() {
  puts("foo");
}

void schulung::baz() {
  deeper::foobar();
}

void schulung::deeper::foobar() {
  foo();
  puts("bar");
}
```

```
/*** main.cpp ***/

#include "foobar.h"

int main() {
  schulung::baz();
  schulung::deeper::foobar();
  return 0;
}
```

When you define namespaces, think of these 2 things:

1. Project name
2. Project structure.

Header in c++ standard library:

C++ takes the headers of the c standard library in the following pattern:

<xyz.h> → <cxyz>

C libraries in c: #include <stdio.h> → puts, printf (in c we didn't had namespace)

C libraries in c++: #include <cstdio> → std::puts, std::printf

Note: the header of the style <xyz.h> is still supported in c++ for compatibility reasons. But don't use.

Struct, union and enum in c++:

Remember in c, we had to use typedef for struct/union/enum XYZ with typedef, to use XYZ alone as the name of the datatype.

```
struct Point {
  double x;
  double y;
};
// Nutzbar als
struct Point p;
```

VS.

```
typedef struct Point {
  double x;
  double y;
} Point ;
// Nutzbar als
Point p;
```

In c++ you don't have to do that:

```
struct Point {
  double x;
  double y;
};
// Nutzbar als
Point p;
```

Extra:

From the slide: (I didn’t get it)

Durch Vorangestelltes :: kann ein
Name absolut angegeben und auf den
globalen Namespace verwiesen werden.
Das ist relativ selten nötig. See the example:

```cpp
#include <stdio.h>

namespace schulung {

void foo() {
  ::puts("foo");
}

namespace deeper {

void foobar() {
  ::schulung::foo();
  ::puts("bar");
}

} // Ende Namespace "deeper"

void baz() {
  ::schulung::deeper::foobar();
}

} // Ende Namespace "schulung"

int main() {
  ::schulung::baz();
  ::schulung::deeper::foobar();
  return 0;
}
```