# The C/C++ preprocessor

P. Ronchese
Dipartimento di Fisica e Astronomia “G.Galilei”

Università di Padova

“Object oriented programming and C++” course

## Process before compilation

Lines starting with `"#"` (no `";"` at the end) contain commands executed before the actual compilation begin: “preprocess”

- Commands useful to:
  - avoid code duplication,
  - add flexibility to compile in different environments,
  - ease debugging.
- Powerful, but sometimes tricky

The code actually seen by the compiler after preprocessing can be seen by issuing the command:

```
c++ -E -o file.pp file.cc
```

# Preprocessor commands

- `#include "file"` , `#include <file>` : the specified *file* is simply included
  - files are looked for in several places
  - different places sequences in the search with the two forms `""` and `<>`
  - adding the option `-I path` the specified *path* is added to the sequence of places to search
- `#define X Y` : replaces `X` with `Y` (“macro”)
- `#undef X` : removes the `X` definition
- `#ifdef X` , `#ifndef X` : process/compile the following (until `#endif`) only if `X` is defined or not
- `#if X==Y` , `#elif X==Z` , `#else` : process/compile the following (until `#endif`) only if `X` is defined equal to `Z` (with `Z` a constant integer)
- `#error "message"` : write the message and terminate the compilation

## Translation unit

A translation unit is the basic unit of compilation in C++.
It consists of the contents of a single source file,
plus the contents of any header files
directly or indirectly included by it,
minus those lines that were ignored
using conditional preprocessing statements.

A single translation unit can be compiled
into an object file, library, or executable program.

## Conditional inclusion

Functions (and not only fuctions) must be declared before being used:

- their declarations are preferably put into dedicated files (“header file”) included by files using them,
- some declarations can be present only once,
- but with several nested `#include` it could happen that an header file is included more than once

```
#ifndef f_h // include guard
#define f_h
int f(int i);
#endif
```

Use `#ifndef` to ensure single compilation

# Preprocessor macros

The C/C++ preprocessor allows the definition of “macros” with arguments, that can be used to mimic functions (with important differences)

```
#define SQUARE(X) X*X
#define PRINT(X) std::cout << #X << " = "
                           <<  X << std::endl

  ...
  int i=12;
  int j=SQUARE(i);
  PRINT(j);
  ...
```

- The argument is simply replaced, e.g. `SQUARE(i)` is replaced by `i*i`
- The argument following a `#` is replaced by a string, e.g. `PRINT(j)` is replaced by `std::cout << "j" << " = " << j << std::endl`

# Macro pitfalls

The code replacement produce a lot of pitfalls!

- The operation priority can be broken:
  - `SQUARE(x+y)` becomes `x+y*x+y` , i.e. `x+(y*x)+y` ,
  - quite different from `(x+y)*(x+y)` (expected).
- Composition of several operations with brackets `{...}` in a macro can break flux control:
  - `if(...) COMPOSITE(i); else COMPOSITE(j);` becomes
    `if(...) {...}; else {...};` ,
  - the first semicolon breaks the
    `if(...) {...} else {...}` syntax
- Workarounds do exist, but they make the code obscure and tricky

Use macros at your own risk, or, better,
do not use them at all unless you're more than experienced.

# Predefined macros

- Some macros are predefined:
  - `__FILE__` : file name
  - `__LINE__` : line number
  - `__DATE__` : compilation date
  - `__TIME__` : compilation time
  - `__cplusplus` : non-zero for C++ compiler
- Macros can be defined in the compilation command:
  - `c++ -D X=Y` has the same effect as `#define X Y`,
  - useful to have different versions in the same file