C++

Separate compilation

Single file compilation

```
// X.cpp
#include <iostream>
class X { public: int x; };
int main (void) {
 X x1;
 x1.x = 1;
 std::cout << x1.x << std::endl;
}
```

To compile:
$ g++ X.cpp

It creates a binary: “a.out” (default name)

To execute:
$ ./a.out

Other possibility:
$ g++ -o X X.cpp

It creates a binary with the name “X”

To execute:
$ ./X

Header files

- In general in C++ header files (.h) contains the class interface, while the implementation is in .cpp files
- Example:

INTERFACE

```
// Date.h
class Date {
 public:
  Date();
  void set(int m, int d, int y);
  void print();
 private:
  int month, day, year;
};
```

IMPLEMENTATION

```
// Date.cpp
#include <iostream>
#include “Date.h”

Date::Date() {}

Void Date::set (int m, int d, int y) {
 month = m; day = d; year = y;
}

Void Date::print () {
  std::cout << month << “ / “ << day
            << “ / “ << year << std::endl;
}
```

Separate compilation

```
// main.cpp
# include “Date.h”

int main(void) {
  Date d;
  d.set(1, 29, 2010);
  d.print();
}
```

Classes or functions using user defined classes need to “import” the interface

To compile:

```
 $ g++ -c Date.cpp
 $ g++ -c main.cpp
 $ g++ -o my_program main.o Date.o
```

Lines 1 and 2 create objects file (“-c” option) Date.o and main.o.
These objects can not run on their own.

Line 3 links the object files together to create an executable program (that can be run)

Why is separate compilation a good idea ?

- Suppose that we change the file “Date.cpp”; for example we change the implementation of print()
- Then we only need to recompile Date.cpp and link (lines 1 and 3) but we do not need to recompile main.cpp.
- On a big project with thousands or hundred of thousands of files where a build can take hours, this is a big win.

Preprocessor directives

- Same as in C, preprocessor directives start with #
- #include reads in the contents of the file:
  #include <iostream>
  #include “my_file.h”
- <> are used for standard header files searched in the standard library directories
- “” are used for user-defined header files searched in the current directory
- It is possible to specify the location to search for header files with the option “-I” of the compiler:
  $ g++ test.cpp -I /home/user_a/include

Preprocessor directives

- Since #include may be nested, the same header file can be included more than once
- Use conditional directives to limit the inclusion to exactly one time:

```
// Date.h
#ifndef DATE_H
#define DATE_H
// rest of the class declaration
#endif // DATE_H
```