Biên dịch
Biên dịch riêng rẽ

Lập trình hướng đối tượng

Biên dịch

- Chỉ hướng dẫn biên dịch trong môi trường Unix, sinh viên tự tìm hiểu đối với các môi trường lập trình khác.
- Ta sẽ sử dụng **g++** để dịch các chương trình C++.

```
g++ foo.cpp
```

- biên dịch **foo.cpp** cho kết quả là file chạy được **a.out**

```
g++ -o foo foo.cpp
```

- biên dịch **foo.cpp** cho kết quả là file chạy được **foo**

Biên dịch riêng rẽ

- VD: biên dịch chương trình **`program.cpp`** trong đó sử dụng một lớp có tên **`Picture`** để thao tác các hình vẽ
- Nên lưu phần cài đặt của lớp **`Picture`** trong một file riêng, chẳng hạn **`picture.cpp`**, để:
  - tạo thuận lợi cho việc sử dụng lớp này trong một ứng dụng khác
  - hai lập trình viên có thể dễ dàng cùng làm việc: một người cài đặt lớp **`Picture`**, người kia viết chương trình chính **`program.cpp`**
  - khi chương trình thay đổi, chỉ cần dịch lại file **`program.cpp`**, như vậy, quá trình biên dịch nhanh hơn. Đối với các chương trình lớn, điều này tạo sự khác biệt rất lớn.
- *Chú ý*: Theo thông lệ, các file chương trình C++ thường có kiểu mở rộng ".cpp", ".cc", ".C", hoặc ".cxx".

File header của lớp: ".h"

- Nếu ta không muốn người viết **`program.cpp`** biết chi tiết của lớp **`Picture`** (vì đó có thể là bí mật thương mại), ta cần tách giao diện của lớp (phần khai báo) ra khỏi cài đặt của lớp.
- Mặt khác, để có thể biên dịch được, chương trình chính **`program.cpp`** cũng cần biết về định nghĩa của lớp **`Picture`** và các phương thức của lớp đó.
- Giải pháp là mô tả lớp **`Picture`** tại hai file
  - picture.h các định nghĩa và khai báo (giao diện)
  - picture.cpp cài đặt

File header của lớp: ".h"

File chứa cài đặt

```
/* picture.h */
class Picture
{
  //...
  Picture* frame(const Picture&);
}
```

```
/* picture.cpp */
#include "picture.h"
Picture* Picture::frame (const Picture& x)
{
   //mã để đóng khung một hình ảnh
}
```

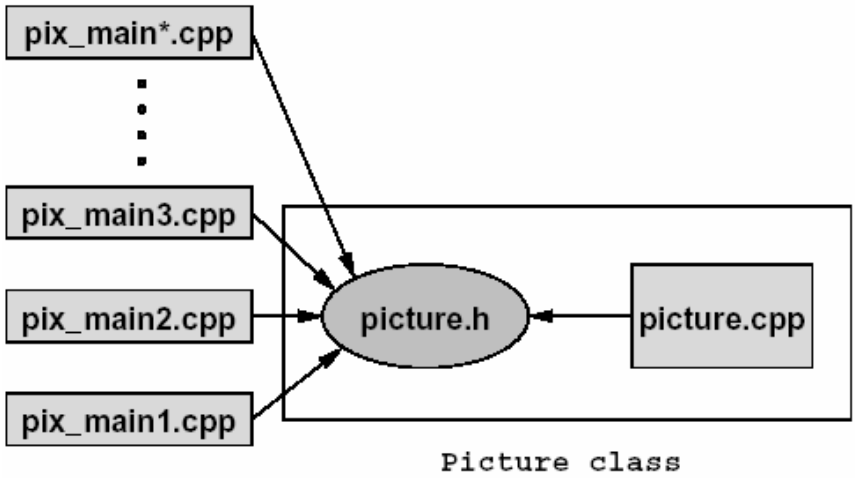
File header chứa giao diện

```
/* program.cpp */
#include "picture.h"
int main()
{
   //thao tác các hình ảnh
}
```

Client/user

File header của lớp: ".h"

Như vậy, ta có thể viết nhiều chương trình sử dụng lớp **`Picture`** có sẵn một cách tiện lợi



Biên dịch riêng rẽ

- biên dịch chương trình như sau:

```
1> g++ -c picture.cpp
2> g++ -c program.cpp
3> g++ -o program program.o picture.o
```

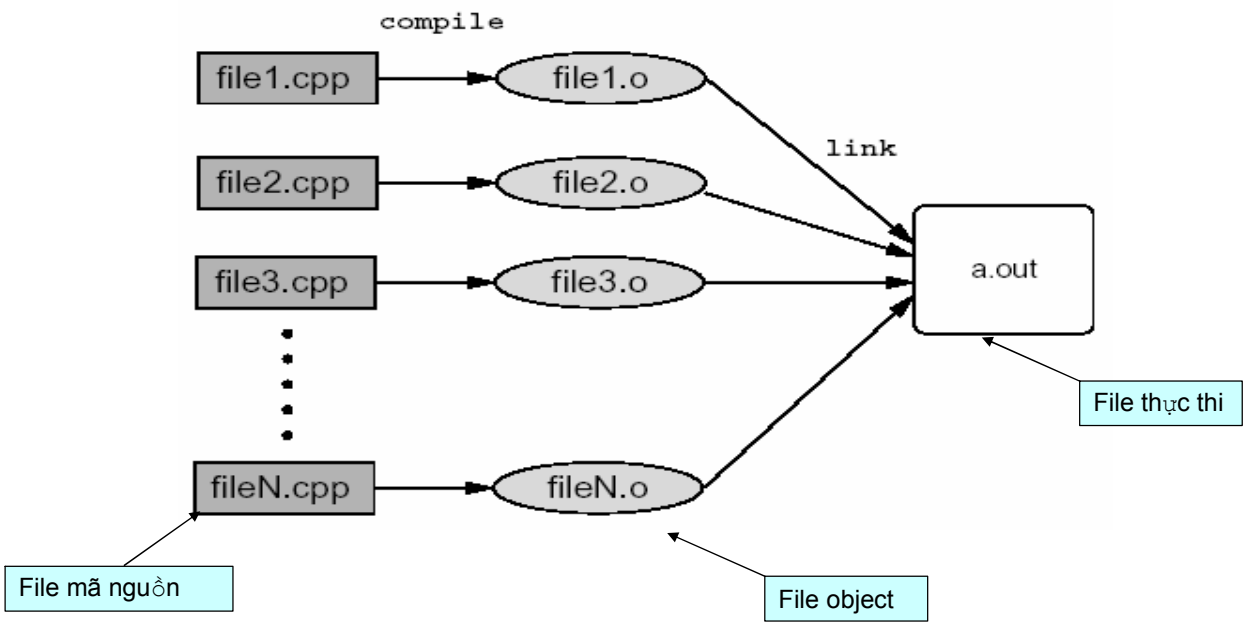
  - khóa chuyển **-c** tại dòng 1 và 2 tạo các object file **program.o** và **picture.o**. Dòng 3 tạo file chạy được có tên **program** với khóa chuyển **-o** bằng cách liên kết các object file với nhau.

- Hoặc

```
1> g++ -c picture.cpp
2> g++ -o program program.cpp picture.o
```

- Nếu **program.cpp** bị thay đổi nhưng **Picture** vẫn giữ nguyên, thì khi biên dịch lại, dòng 1 là không cần thiết.

Liên kết object file



Các định hướng tiền xử lý

- Các định hướng tiền xử lý là các lệnh có tính năng đặc biệt
- Được thực hiện bởi trình tiền xử lý trước khi mã nguồn được biên dịch.
- Trong C++, các định hướng tiền xử lý bắt đầu bằng một dấu #
- #include
- #define, #ifndef, #endif

Định hướng tiền xử lý #include

- Định hướng **`#include`** đọc nội dung của file được nêu tên vào nơi đặt định hướng

```
#include <standard_file.h>
#include "my_file.h"
```

- Cặp ngoặc nhọn < > dùng cho các file header chuẩn được tìm kiếm trong các thư mục thư viện chuẩn.
- Cặp dấu nháy " " dùng cho các file header của người dùng, sẽ được tìm kiếm trước hết trong thư mục hiện tại.
  - Có thể dùng khoá chuyển **`-I (g++ -I)`** để thay đổi đường dẫn tìm kiếm. Ví dụ:

```
g++ program.cpp -I/home/tmct/my_include/
```

trong đó, `/home/tmct/my_include/` là đường dẫn đầy đủ đến các thư mục chứa các file .h cần tìm

Các thư viện

- Để tạo một file thực thi (executable file), trình liên kết (linker) cần kết nối mã của các hàm được khai báo trong các file header chuẩn C++ (iostream.h, string.h, v.v..) Các đoạn mã tương ứng có thể được tìm thấy trong các thư viện chuẩn C++
- Một thư viện là một tập hợp các object file.
- Trình liên kết lựa chọn mã object từ các thư viện chứa định nghĩa các hàm được sử dụng trong các file chương trình và kết nối chúng vào file thực thi (executable file).
- Một số thư viện được trình liên kết C++ tự động sử dụng, chẳng hạn thư viện chuẩn C++. Các thư viện khác phải được chỉ rõ trong quá trình liên kết bằng khoá chuyển **`-l`**. Ví dụ, trong một số môi trường lập trình, cần lệnh sau để liên kết với thư viện toán học chuẩn **`libm.a`**

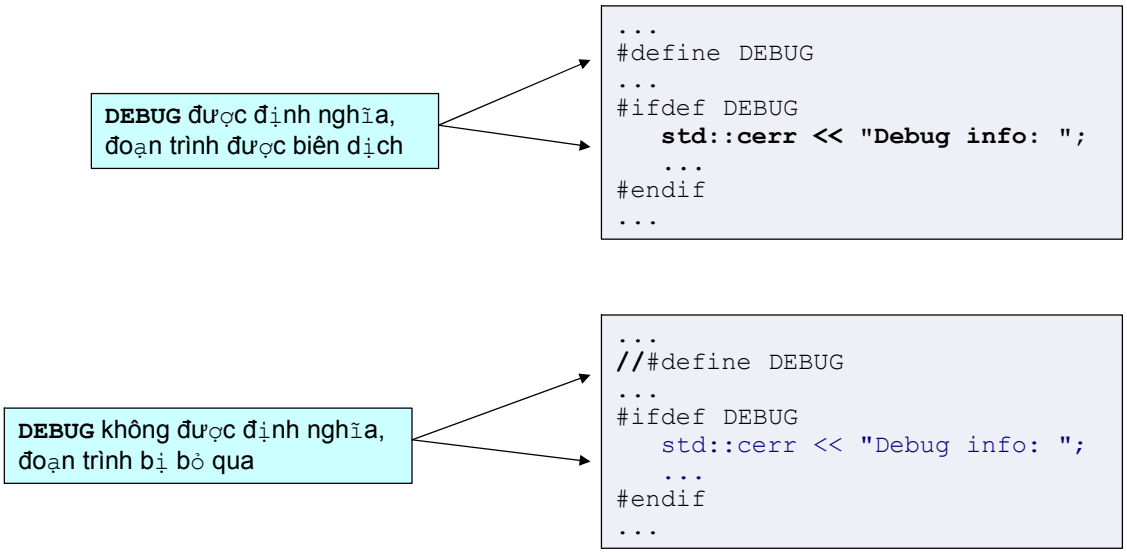
```
g++ -o myprog myprog.o -lm
```


#define, #ifdef, #ifndef, #endif

- **`#define`** định nghĩa một định danh
  - `#define MAX 100 // từ đây, MAX sẽ có giá trị 100`
  - `#define DEBUG // định nghĩa DEBUG`
- **`#ifdef`** định hướng điều kiện "nếu đã định nghĩa" (if defined)
  - `#ifdef DEBUG // nếu DEBUG đã được định nghĩa`
- **`#ifndef`** định hướng điều kiện "nếu chưa định nghĩa" (if not defined)
  - `#ifndef DEBUG // nếu DEBUG chưa được định nghĩa`
- **`#endif`** kết thúc khối mở đầu bằng **`#ifndef`** hoặc **`#ifdef`** gần nhất
  - nếu điều kiện tại định hướng mở đầu khối thỏa mãn thì biên dịch đoạn lệnh nằm trong khối

#define, #ifdef, #ifndef, #endif

- Ví dụ sử dụng



#define, #ifdef, #ifndef, #endif

```
/* program.h */
#include "b.h"
#include "c.h"
…
```

```
/* b.h */
#include "a.h"
#include "d.h"
…
```

```
/* c.h */
#include "a.h"
#include "e.h"
…
```

- Do các định hướng #include có thể lồng nhau, một file header có thể được kết nối hai lần. Hậu quả là
  - file đó được xử lý nhiều lần **à** tốn thời gian,
  - các hằng, macro, kiểu dữ liệu, nguyên mẫu hàm… được khai báo nhiều lần **à** lỗi biên dịch.
- Do vậy, ta cần các định hướng điều kiện (conditional directive) trong mọi file header

```
#ifndef PICTURE_H
#define PICTURE_H
// các khai báo đối tượng, định nghĩa lớp, hàm…
#endif      //PICTURE_H
```