

C editing with VIM HOWTO

written by Siddharth Heroor

<http://www.tldp.org/HOWTO/C-editing-with-VIM-HOWTO/index.html>
version 1.0

Dịch bởi: Hoang Tran
hoang.tran@vnoss.org

Tài liệu này giới thiệu cách sử dụng Vi/ViM để viết các file mã ngôn ngữ C và các ngôn ngữ khác có cú pháp tương tự như C++ hay Java.

Mục lục

| | | |
|----------|--|-----------|
| 1 | Giới thiệu | 3 |
| 2 | Dịch chuyển | 3 |
| 2.1 | w, e và b | 3 |
| 2.2 | {, }, [[và]] | 3 |
| 2.3 | % | 5 |
| 3 | Nhảy ngẫu nhiên trong file mã nguồn C | 6 |
| 3.1 | ctags | 6 |
| 3.2 | marks | 7 |
| 3.3 | gd | 8 |
| 4 | Tự động hoàn thành các từ | 8 |
| 5 | Tự động thay đổi định dạng | 10 |
| 5.1 | Giới hạn chiều rộng | 10 |
| 5.2 | Tự động căn chỉnh mã | 11 |
| 5.3 | Chú giải | 11 |
| 6 | Soạn thảo nhiều file | 12 |
| 7 | Sửa lỗi nhanh | 12 |
| 8 | Copyright | 14 |
| 9 | References | 14 |

1 Giới thiệu

Mục đích của tài liệu này là để giới thiệu với những người sử dụng VIM trình độ cơ bản những chức năng soạn thảo những file mã nguồn C trong VIM. Tài liệu này giới thiệu một vài lệnh và các keystrokes làm tăng hiệu suất của lập trình viên khi sử dụng VIM để lập trình C.

Trong phạm vi của tài liệu, chúng ta sẽ diễn tả việc soạn thảo chương trình C với VIM như thế nào. Tuy nhiên thì hầu hết những gì được diễn tả cũng có thể hoạt động với VI. Thêm vào đó những gì được đề cập ở đây về việc thảo những file mã nguồn C thì có thể thích hợp nhiều hoặc ít hơn với C++, Java hay những ngôn ngữ tương tự khác.

2 Dịch chuyển

2.1 w, e và b

Chúng ta có thể sử dụng các phím w, e và b để dịch chuyển trong file. VIM có khả năng nhận ra các thẻ từ (tokens) khác nhau trong các lệnh C.

Hãy xem xét đoạn mã C sau

```
...
if( ( NULL == x ) && y > z )
...
```

Giả sử rằng con trỏ đang nằm ở vị trí đầu tiên của câu lệnh `if`. Bằng việc ấn w một lần thì con trỏ sẽ nhảy đến dấu (đầu tiên. Ấn tiếp w thì con trỏ sẽ dịch chuyển đến `NULL`. Tiếp theo con trỏ sẽ dịch chuyển đến `==`. Các phím w tiếp theo sẽ đưa bạn đến `x ...) ... && ... y ... > ... z ...` và cuối cùng là `) ...`

e thì tương tự như chỉ có điều rằng nó sẽ bạn tới vị trí cuối cùng của từ hiện thời bạn đang đứng và không phải là chữ cái đầu tiên của từ tiếp theo.

b hoạt động ngược hẳn với w. Nó dịch chuyển con trỏ theo chiều ngược lại. Do đó bạn dịch chuyển ngược lại sử dụng phím b.

2.2 {, }, [[và]]

Phím { và } được sử dụng để chuyển từ đoạn (paragraph) này sang đoạn khác. Khi soạn thảo file C, những phím này có một ý nghĩa hơi khác một chút. Sau đây là một đoạn có nhiều dòng được chia cắt bởi một dòng trống.

Ví dụ:

```

...
C-statement;
/* Comment */
...
/* Next Set of C-statements */
...

```

Đoạn mã trên chỉ ra có hai đoạn. Rất dễ dàng có thể dịch chuyển từ của một đoạn đến đoạn kia bằng việc sử dụng các phím { và }. { sẽ đưa con trỏ vào đoạn trên và } sẽ đưa con trỏ tới đoạn dưới.

Rất nhiều người có phong cách viết mã khi mà các nhóm lệnh cùng logic được nhóm lại với nhau và cách với nhóm lệnh khác nhiều hơn một dòng trống.

Ví dụ:

```

void function1()
{
    /* Declarations */
    int x;
    char y;
    double z;

    /* Some code */
    x = 1;
    y = 'a';
    z = 1.2;

    /* Some more code */
    x++;
    y++;
    z++;
}

```

Phím { và } rất hữu dụng trong những trường hợp này. Rất dễ dàng để dịch chuyển từ đoạn này sang đoạn khác.

Một tập các phím khác cũng rất hữu dụng là phím [[và]]. Những phím này cho phép chúng ta nhảy tới trước { hoặc sau } ở cột đầu tiên.

Ví dụ

```

void foo()
{
    /* Some C-statements */
}
void bar()
{
    /* Some other C-statements */
}

```

Giả sử bạn đang sửa hàm foo() và bây giờ bạn muốn chuyển sang sửa

hàm `bar()`. Chỉ đơn giản là ấn `]]` và con trỏ sẽ đưa bạn đến `{` của hàm `bar()`. Ngược lại thì khác biệt chút ít. Nếu bạn đang ở giữa hàm `bar()` và bạn ấn `[[`, con trỏ sẽ đưa bạn tới vị trí `{` phía trước, có nghĩa là vị trí bắt đầu của bản thân hàm `bar()`. Ấn tiếp `[[` thì con trỏ sẽ dịch chuyển tới đầu hàm `foo()`. Số lần ấn phím có thể giảm đi bằng việc ấn `2[[` để đưa con trỏ về vị trí bắt đầu của hàm phía trước.

Một tập tổ hợp phím khác tương tự là `]]` và `[]`. `]]` đưa con trỏ tới vị trí `}` tiếp theo ở cột đầu tiên. Nếu bạn đang sửa hàm `foo()` và muốn tới vị trí cuối cùng của hàm `foo()` thì `]]` sẽ đưa bạn tới đó. Tương tự nếu bạn đang sửa hàm `bar()` và muốn tới vị trí cuối cùng của hàm `foo()` thì `[]` sẽ đưa con trỏ tới đó.

Một cách để nhớ các phím này là chia rẽ chúng ra. Phím đầu tiên sẽ chỉ ra là sẽ dịch chuyển lên hay xuống. `[` sẽ dịch chuyển lên và `]` sẽ dịch chuyển xuống. Phím tiếp theo chỉ ra kiểu dấu ngoặc móc thích hợp. Nếu nó giống với phím trước đó thì con trỏ sẽ chuyển đến `{`. Nếu nó khác, thì con trỏ sẽ chuyển đến `}`.

Chú ý rằng các phím `]]`, `[[`, `[[`, và `[]` là chỉ tìm các kiểu dấu ngoặc móc thích hợp ở cột đầu tiên (ký tự đầu tiên trong hàng). Nếu chúng ta muốn chúng thích hợp với tất các dấu ngoặc móc cho dù nó ở cột đầu tiên hay không là điều không thể. Trong tài liệu về VIM có một cách khác phục điều này. Đó là hãy ánh xạ (map) các phím để tìm ra cái dấu ngoặc móc thích hợp. Không cần mất nhiều thời gian vào tìm hiểu kỹ thuật ánh xạ (mapping), ánh xạ gợi ý là:

```
:map [[ ?{<CTRL-VCTRL-M>w99 [{
:map ]] /}<CTRL-VCTRL-M>b99}]
:map [[ j0[[%/{<CTRL-VCTRL-M>
:map [] k$][%?<CTRL-VCTRL-M>
```

2.3 %

Phím `%` được sử dụng để tìm các ký tự tương ứng với ký tự dưới trỏ. Ký tự dưới con trỏ có thể là dấu ngoặc tròn, dấu ngoặc móc hoặc dấu ngoặc vuông. Bằng việc ấn phím `%` con trỏ sẽ nhảy đến dấu ngoặc tương ứng.

Một điểm khác là phím `%` cũng có thể được sử dụng để ánh xạ với `#if`, `#ifdef`, `#else` `#elif` và `#endif`.

Phím này rất hữu dụng trong việc kiểm tra tính hợp lệ của đoạn mã đang viết. Ví dụ:

```
...
if((x==y) && ((z==a) || (y>x)))
...
```

Hãy thử với đoạn mã trên bằng việc kiểm tra sự chính xác của các dấu ngoặc. Phím % sẽ giúp nhảy từ dấu (tới dấu) tương ứng và ngược lại. Do đó có thể tìm dấu mở ngoặc tương ứng với dấu đóng ngoặc để kiểm tra tính hợp lệ của đoạn mã.

Tương tự thì % cũng có thể để nhảy từ { tới } tương ứng.

3 Nhảy ngẫu nhiên trong file mã nguồn C

3.1 ctags

Một thẻ (tag) là một kiểu giữ vị trí. Các thẻ rất hữu dụng trong việc hiểu và soạn thảo mã nguồn C. Các thẻ là tập các thẻ đánh dấu cho mỗi hàm trong file C. Nó rất có ích trong việc nhảy đến các định nghĩa của các hàm từ nơi hàm đó được gọi, và nhảy ngược lại. Hãy xem ví dụ sau đây

```
...
foo()
{
    ...
    bar();
    ...
}
bar()
{
    ...
}
```

Giả sử rằng bạn đang sửa ở hàm foo() và đang ở lời gọi hàm bar(). Bây giờ bạn muốn xem hàm bar() đã làm gì. Có một cách là bạn hãy sử dụng các thẻ (tags). Bạn có thể nhảy tới định nghĩa của hàm bar và nhảy ngược lại nơi đã gọi hàm bar(). Nếu muốn bạn có thể nhảy tới hàm khác được gọi bên trong hàm bar và nhảy ngược lại.

Để sử dụng tags, trước hết bạn phải chạy chương trình ctags để tạo tags cho toàn bộ các file mã nguồn. Nó sẽ tạo ra một file là tags. File này sẽ chứa tất cả các con trỏ tới các định nghĩa các hàm và được sử dụng bởi VIM để giúp bạn nhảy tới các định nghĩa các hàm.

Tổ hợp phím giúp cho việc nhảy tới và lui đó là **CTRL-J** và **CTRL-T**. Bằng việc ấn **CTRL-J** trong hàm foo() tại nơi hàm bar() được gọi sẽ đưa con trỏ tới điểm bắt đầu của hàm bar(). Để nhảy ngược lại từ hàm bar() tới foo() chỉ cần ấn **CTRL-T**.

ctags được gọi bằng cách

```
$ ctags options file(s)
```

Để tạo file tags từ tất cả các *.c files trong thư mục hiện tại, bạn chỉ cần

```
$ ctags *.c
```

Trong trường hợp cây thư mục mã nguồn chứa các file C nằm trong các thư mục con khác nhau, chúng ta có thể gọi ctags ở thư mục gốc của cây thư mục với tham số -R và một file tags chứa các thẻ tới tất cả các hàm trong thư mục sẽ được tạo ra. Ví dụ

```
$ ctags -R *.c
```

Có rất nhiều lựa chọn khác nhau để sử dụng với ctags. Những lựa chọn này được giải thích trong file trợ giúp cho ctags. (\$ man ctags)

3.2 marks

Đánh dấu cũng là cách lưu trữ vị trí giống như các thẻ (tags). Tuy nhiên, các điểm đánh dấu thì có thể được thiết lập ở bất kỳ vị trí nào trong file và không có giới hạn chỉ trong các hàm (functions), enums, ... Thêm nữa cách đánh dấu thì được thiết lập bằng tay bởi người sử dụng.

Khi thiết lập các vị trí đánh dấu thì không có dấu hiệu nhìn thấy nào ở đó cả. Một điểm đánh dấu trong một file chỉ được đơn giản được nhớ bởi VIM (và trong não bạn nữa). Hãy xem xét đoạn mã dưới đây

```
void foo()
{
    int x, y;
    x = 0;
    y = 1;
    x++;
    y++;
    if (x != y)
        x = y;
    y = x;
}
```

Giả sử bạn đang sửa dòng x++; và bạn muốn trở lại nó sau khi sửa các dòng khác nữa. Bạn có thể thiết lập một điểm đánh dấu ở dòng đó bằng phím **m** và trở lại dòng đó sau đó bằng ấn **”**.

VIM cho phép bạn có thể thiết lập nhiều hơn một vị trí đánh dấu. Những điểm đánh dấu được lưu trữ trong các thanh ghi a-z, A-Z và 1-0. Để thiết lập một vị trí đánh dấu được lưu trữ trong cùng một thanh ghi (giả sử là j), đơn giản chỉ cần ấn **mj**. Để trở lại vị trí đánh dấu thì chúng ta ấn **j**.

Nhiều vị trí đánh dấu thực sự hữu ích trong việc dịch chuyển tới và lui. Lấy một ví dụ tương tự, bạn có thể muốn đánh dấu một điểm ở x++; và

một điểm khác tại $y=x$; và sau đó nhảy giữa chúng hoặc là một nơi khác sau đó quay trở lại.

Các điểm đánh dấu có thể mở rộng thông qua các file khác nhau. Để sử dụng các điểm đánh dấu này chúng ta phải sử dụng các thanh ghi có chữ hoa, ví dụ A-Z. Các thanh ghi chữ thường chỉ hoạt động trong một file và không mở rộng giữa các file. Điều đó có nghĩa là nếu bạn thiết lập một điểm đánh dấu trong file `foo.c` với thanh ghi "a" và sau đó chuyển sang file khác và ấn 'a, con trỏ sẽ không thể nhảy ngược lại vị trí trước. Nếu bạn muốn đánh dấu một điểm mà có thể đưa bạn tới những file khác nhau thì bạn cần sử dụng một thanh ghi chữ hoa. Ví dụ, sử dụng `mA` thanh vì `ma`. Tôi sẽ nói về việc soạn thảo nhiều file ở một mục khác.

3.3 gd

Hãy xem xét đoạn mã dưới đây

```
struct X x;
void foo()
{
    struct Y y;
    struct Z z;
    ...
    /* Lots of lines later */
    x.bar();
    y.bar();
    z.bar();
}
```

Vì một vài lý do bạn quên rằng `y` và `z` là gì và bạn muốn nhảy tới nơi khai báo chúng một cách nhanh chóng. Một cách là thực hiện việc search ngược chiều cho `y` và `z`. VIM đề nghị một giải pháp khác đơn giản và nhanh hơn nhiều. Tổ hợp phím `gd` viết tắt của Goto Declaration có nghĩa là "nhảy tới nơi khai báo". Khi vị trí con trỏ ở "y" bạn ấn `gd` thì con trỏ sẽ đưa bạn tới nơi khai báo biến `y: struct Y y;`.

Tổ hợp phím tương tự là `gD`. Nó sẽ đưa bạn tới nơi khai báo biến toàn cục của biến nằm dưới con trỏ. Do đó nếu bạn muốn nhảy tới nơi khai báo biến `x`, bạn chỉ cần thực hiện ấn `gD` và con trỏ sẽ nhảy tới nơi khai báo biến `X: struct X x;`

4 Tự động hoàn thành các từ

Hãy xem xét đoạn mã dưới đây

```

void A_Very_Long_Function_Name ()
{
    ...
}
short A_Very_Long_Variable_Name;
void Another_Function ()
{
    ...
    A_Very_Long_Function_Name ();
    ...
}

```

Hàm `A_Very_Long_Function_Name()` có thể rất là khó chịu để gõ đi gõ lại. Trong khi ở chế độ insert, chúng ta có thể tự động hoàn thành từ đó bằng việc hoặc là search tiến hoặc lui. Trong hàm `Another_Function()`, giả sử bạn đang gõ `A_Very...` và gõ **CTRL-P**. Từ đầu tiên thích hợp sẽ hiển thị trước. Trong trường hợp này có thể là `A_Very_Long_Variable_Name`. Để hoàn thành nó chính xác, bạn gõ **CTRL-P** lần nữa và quá trình tìm kiếm ngược lên tiếp tục tới từ tiếp theo thích hợp là `A_Very_Long_Function_Name`. Khi tìm được từ thích hợp thì bạn có thể tiếp tục gõ. VIM vẫn để ở trong chế độ insert trong suốt quá trình sử lý.

Tương tự với **CTRL-P** là tổ hợp phím **CTRL-N**. Quá trình tìm kiếm sẽ là tìm kiếm tiến (forward) thay vì lui lên (backward). Cả hai tổ hợp phím tiếp tục tìm kiếm cho đến khi nó tới đầu hoặc cuối.

Cả **CTRL-P** và **CTRL-N** là một phần của chế độ được biết đến như là chế độ CTRL-X. Chế độ CTRL-X là một chế độ con của chế độ insert. Do đó bạn có thể vào chế độ này khi bạn đang ở chế độ insert. Để thoát khỏi chế độ CTRL-X, bạn có thể gõ bất kỳ phím nào khác CTRL-X, CTRL-P và CTRL-N. Một khi bạn rời chế độ CTRL-X, bạn trở về chế độ insert.

Chế độ CTRL-X cho phép bạn tự động hoàn thành từ theo nhiều cách khác nhau. Một trong số chúng là tự động hoàn thành tên file. Nó đặc biệt hữu dụng khi bạn phải thêm vào các file tiêu đề (header files). Sử dụng chế độ CTRL-X, bạn có thể thêm `foo.h` bằng cách sử dụng mô hình sau.

```
#include "f CTRL-X CTRL-F"
```

Đó là CTRL-X CTRL-F. Nó có vẻ hơi giống emacs ;-). Có những thứ khác mà bạn có thể làm việc trong chế độ CTRL-X. Một trong số chúng là sử dụng từ điển các từ. Từ điển cho phép chỉ định một file chứa một danh sách các từ mà được sử dụng trong việc tự động hoàn thành từ. Mặc định thì lựa chọn từ điển không được thiết lập. Nó có thể thiết lập bằng lệnh `:set dictionary=file`. Bạn có thể đưa các từ khóa, các kiểu và các định nghĩa trong C vào trong file từ điển. Lập trình viên C++ và Java cũng có thể thích thú với việc tên lớp.

Định dạng của file từ điển rất đơn giản. Chỉ cần đưa một từ bạn muốn vào một hàng. Do đó một file từ điển C có thể trông như thế này.

```
break
case
continue
default
define
do
double
else
enum
float
for
goto
if
ifdef
ifndef
include
int
pragma
return
struct
switch
typedef
void
while
```

Để sử dụng từ điển này, bạn cần ấn **CTRL-X CTRL-K**. Quá trình hoàn thành tương tự với tổ hợp phím **CTRL-P** và **CTRL-N**. Do đó thay vì gõ "typedef" tất cả các chữ, chúng ta gõ **t CTRL-X CTRL-K** cho đến khi nó được hoàn thành.

5 Tự động thay đổi định dạng

5.1 Giới hạn chiều rộng

Đôi khi chúng ta phải giới hạn chiều rộng là 80 hay 75 hay gì đó. Điều đó có thể được thiết lập dễ dàng bằng lệnh

```
:set textwidth=80
```

Để làm việc này tự động bạn chỉ cần đưa lệnh đó vào trong file `.vimrc` của bạn.

Ngoài chiều rộng đoạn mã bạn gõ, bạn có thể muốn vùng văn bản được bao ở một cột nào đó. Sở dĩ có những lựa chọn này bởi vì ở terminal thường sử dụng chúng. Lệnh thích hợp cho trường hợp này là

```
:set wrapwidth=60
```

Lệnh trên làm cho đoạn văn bản bị bao ở 60 cột.

5.2 Tự động căn chỉnh mã

Khi lập trình trong C chúng ta thường phải căn lề các khối lệnh cho dễ đọc. Để làm việc này một cách tự động, VIM có một lựa chọn là `cindent`. Để thiết lập nó chỉ cần sử dụng lệnh

```
:set cindent
```

Khi sử dụng `cindent` thì đoạn mã sẽ tự động căn chỉnh làm cho nó đẹp hơn. Hãy thêm lệnh đó vào trong file `.vimrc` của bạn để nó tự động chạy khi khởi động VIM.

5.3 Chú giải

VIM cũng cho phép bạn tự động định dạng các khối chú giải (comments). Bạn có thể chia chú giải thành 3 phần: phần đầu, phần giữa và phần cuối. Ví dụ, phong cách viết mã của bạn có thể sẽ đòi hỏi phải viết chú giải theo phong cách sau

```
/*  
 * This is the comment  
*/
```

Trong trường hợp này, những lệnh sau có thể sử dụng

```
:set comments=sl:/*,mb:*,elx:*/
```

Hãy để tôi giải mã lệnh trên cho bạn. Lệnh đó có ba phần. Phần đầu là `sl:/*`. Nó nói với VIM rằng ba đoạn comments sẽ bắt đầu với `/*`. Phần tiếp theo cho biết rằng phần giữa của phần chú giải sẽ bắt đầu với `*`. Phần cuối cùng của lệnh trên cho biết 2 điều. Một là lệnh trên sẽ kết thúc với `*/` và nó sẽ tự động hoàn thành phần chú giải khi bạn chỉ ấn `/`.

Lấy một ví dụ khác. Giả sử bạn cần chú giải theo cách thức như sau

```
/*  
** This is the comment  
*/
```

Trong trường hợp này bạn có thể sử dụng lệnh sau

```
:set comments=sl:/*,mb:**,elx:*
```

để thêm chú giải vào sau khi gõ `/*` và gõ enter. Dòng tiếp theo sẽ tự động chứa `**`. Sau khi bạn kết thúc việc chú giải, bạn chỉ cần gõ enter lần nữa và `**` sẽ được thêm vào. Tuy nhiên vào cuối đoạn chú giải, bạn muốn chỉ có `*/` và không phải `**/`. VIM khá là thông minh ở đây. Bạn không cần xóa ký tự `*` cuối cùng và thay thế ni bởi `/`. Thay vì đó, chỉ cần ấn `/` và VIM sẽ nhận ra nó là việc kết thúc đoạn chú giải và sẽ tự động thay thế `**` thành `*/`.

Hãy tham khảo thêm thông tin bằng lệnh `:h comments`

6 Soạn thảo nhiều file

Thường thì chúng ta cần phải soạn thảo nhiều hơn một file một lúc. Ví dụ như phải đồng thời soạn thảo cả các header file và source file. Để soạn thảo nhiều hơn một file ở một thời điểm, hãy gọi lệnh VIM sau

```
$ vim file1 file2 ...
```

Và bây giờ bạn sửa file đầu tiên và chuyển đến file tiếp theo bằng lệnh

```
:n
```

Bạn có thể nhảy về trước bằng lệnh

```
:e#
```

Sẽ rất hữu dụng khi đang viết mã nếu bạn có thể nhìn cả hai file ở cùng một lúc và có thể chuyển đổi giữa chúng. Nói một cách khác, sẽ rất tuyệt nếu màn hình được chia ra và bạn có thể nhìn header file ở phía trên và source file ở phía dưới. VIM có lệnh để chia màn hình ra như vậy. Hãy sử dụng lệnh `:split`

Cùng một file sẽ được hiển thị ở cả hai cửa sổ. Bất kỳ lệnh nào được thực thi sẽ chỉ ảnh hưởng đến cửa sổ nó đang focus. Do đó có thể sửa một file khác ở cửa sổ khác bằng sử dụng lệnh `:e file2`.

Sau khi thực hiện lệnh đó, bạn sẽ nhận ra rằng có 2 files được hiển thị. Một cửa sổ chỉ tới file đầu tiên và cái còn lại hiển thị file thứ hai. Để chuyển đổi giữa các file bạn phải sử dụng tổ hợp phím `CTRL-W CTRL-W`. Để học cách sử dụng chia các cửa sổ, hãy chạy lệnh `help` về chúng.

7 Sửa lỗi nhanh

Khi lập trình với C, chúng ta hay bắt gặp chu trình sửa-biên dịch-sửa. Nên bạn sẽ muốn sửa file C sử dụng cách mà tôi đã đề cập trước đó, ghi lại file rồi biên dịch mã và nhảy tới những chỗ bị lỗi rồi tiếp tục sửa tiếp. VIM giúp

chúng ta tiết kiệm thời gian ở chu trình đó một chút bằng cách sử dụng ở chế độ gọi là sửa lỗi nhanh (quickfix). Về cơ bản phải lưu lại các lỗi biên dịch trong một file và mở file đó với VIM bằng lệnh

```
$ vim -q compiler_error_file
```

VIM tự động mở file chứa các lỗi và vị trí con trỏ sẽ nằm ở vị trí lỗi đầu tiên.

Có một phím tắt cho chu trình này. Sử dụng lệnh "make", chúng ta có thể tự động biên dịch mã và nhảy tới vị trí đầu tiên bị lỗi. Để thực thi lệnh make hay gõ

```
:make
```

Cơ bản thì lệnh này gọi lệnh make ở shell và nhảy tới vị trí lỗi đầu tiên. Tuy nhiên nếu bạn biên dịch không sử dụng make và biên dịch sử dụng một lệnh khác như cc, thì bạn phải thiết lập một biến gọi là makeprg tới lệnh mà bạn muốn thực thi khi gọi lệnh make. Ví dụ: `:set makeprg=cc foo.c`

Sau khi thiết lập makeprg, bạn có thể chỉ cần gọi lệnh make.

Sau khi bạn sửa lỗi đầu tiên, việc tiếp theo cần làm là nhảy tới chỗ bị lỗi tiếp theo và sửa nó. Lệnh sau đây được sử dụng để nhảy tới lỗi tiếp theo

```
:cn.
```

Để quay trở lại lỗi phía trước bạn có thể sử dụng lệnh `:cN`.

Để diễn tả ví dụ sử dụng cách này hãy xem xét đoạn mã sau

```
#include <stdio.h>

int main()
{
    printf("Hello World\n")
}
```

Bạn có thể nhận ra rằng có một lỗi ở dòng 5. File này được lưu là file test.c và makeprg được thiết lập sử dụng lệnh

```
:set makeprg=gcc\ test.c
```

Tiếp theo lệnh make được thực thi bằng lệnh `:make`. gcc sẽ đưa ra một lỗi và màn hình hiển thị của lệnh make sẽ như sau

```
#include <stdio.h>

int main()
{
    printf("Hello World\n")
}
```

```
:!gcc test.c 2>&1| tee /tmp/vim9821.err
test.c: In function 'main':
test.c:6: parse error before '}'
(2 of 2): parse error before '}'
Press RETURN or enter command to continue
```

Khi ấn RETURN, con trỏ sẽ nhảy tới dòng 6.

Bây giờ, lệnh :cn sẽ đưa con trỏ về dòng 4.

Để quay trở lại lỗi phía trước chúng ta có thể sử dụng lệnh :cN và con trỏ sẽ chuyển lại về dòng 6.

Sau khi sửa lỗi ở dòng 5 và thêm vào "return 1;", chúng ta có thể chạy lại lệnh :make và màn hình hiển thị sẽ như sau

```
#include <stdio.h>

int main()
{
    printf("Hello World\n")
}

:!gcc test.c 2>&1| tee /tmp/vim9822.err

Press RETURN or enter command to continue
```

Đây chỉ là một ví dụ nhỏ. Bạn có thể sử dụng cách thức sửa lỗi nhanh này để giải quyết các vấn đề thời gian biên dịch của bạn và hy vọng rằng nó sẽ làm giảm thời gian của chu trình sửa-biên dịch-sửa.

8 Copyright

Copyright (c) 2000,2001 Siddharth Heroor.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license can be found at <http://www.gnu.org/copyleft/fdl.html>

9 References

You can get more information on VIM and download it at <http://www.vim.org>