# Chỉ Thị Tiền Xử Lý trong C/C++

Chỉ thị tiền xử lý là những chỉ thị cung cấp cho bộ tiền xử lý để xử lý những thông tin trước khi bắt đầu quá trình biên dịch. Tất cả các chỉ thị tiền xử lý đều bắt đầu với với # như #include, #if, #define...

### Chỉ thị tiền xử lý (preprocessor directives)

Chỉ thị tiền xử lý là những chỉ thị cung cấp cho bộ tiền xử lý để xử lý thông tin trước khi bắt đầu quá trình biên dịch.

Tất cả các chỉ thị tiền xử lý đều bắt đầu với với # và không kết thúc bằng dấu chấm phẩy ; khi kết thúc.

Để cho dễ phân biệt chúng ta chia thành 3 nhóm chính đó là:

Chỉ thị chèn tệp (#include).

Chỉ thị định nghĩa cho tên (#define macro).

Chỉ thị biên dịch có điều kiện (#if, #else, #elif, #endif, ...).

#### Chỉ thị chèn tệp (#include)

Ở nhóm này chỉ có một chỉ thị đó là #include. Đây là chỉ thị cho phép chèn nội dung một file khác vào file đang viết.

Cú pháp 1:

#include <file\_name>

Với cú pháp 1 (dùng dấu ngoặc nhọn), bộ tiền xử lý sẽ tìm file\_name có sẵn trong SDK và chèn vào file đang viết, nếu tìm không thấy file\_name sẽ gây ra lỗi. Các file có sẵn trong SDK như stdio.h, math.h, conio.h, ….

Ví dụ:

#include <stdio.h>

Cú pháp 2:

#include "file\_name"

Khi sử dụng cú pháp 2 (dùng dấu nháy đôi), bộ tiền xử lý sẽ tìm file\_name trong các thư mục trên máy tính, khi tìm không thấy thì tiếp tục tìm trong các file có sẵn trong SDK. Nếu tìm được file\_name thì chèn nội dung file\_name vào file đang thao tác, nếu vẫn không tìm thấy file\_name thì sinh ra báo lỗi.

Ví dụ:

#include "Hello.h"

Để hiểu bạn hình dung rõ hơn về cơ chế hoạt động của chỉ thị #include thì bạn theo dõi ví dụ sau đây.

Giả sử file Student.h có nội dung như sau:

struct Student

{

int m\_id;

char\* m\_name;

};

Trong file main.cpp, nếu muốn sử dụng struct Student thì phải #include "Student.h".

// main.cpp

#include "Student.h"

int main()

{

struct Student Nguyen;

return 0;

}

Việc #include "Student.h" giống như việc chép tất cả các đoạn code trong file Student.h vào file main.cpp.

struct Student

{

int m\_id;

char\* m\_name;

};

int main()

{

struct Student Nguyen;

return 0;

}

#### Chỉ thị định nghĩa cho tên (#define macro)

Ở nhóm này gồm các chỉ thị #define, #undef.

##### Chỉ thị #define

###### Chỉ thị #define không có đối số

Cú pháp:

#define identifier replacement-list

Chỉ thị này có tác dụng thay thế tên (identifier) bằng một dãy kí tự sau nó, khi dãy kí tự thay thế quá dài và sang dòng mới thì có thể sử dụng dấu \ vào cuối dòng trước.

Ví dụ:

#define STDIO "stdio.vn" // định nghĩa cho STDIO

Trong hàm main ta thực hiện lệnh sau:

printf(STDIO); // tương đương với printf("stdio.vn");

Phạm vi của tên được định nghĩa bởi #define là lúc từ khi nó được định nghĩa cho đến cuối tệp.

Có thể dùng #define định nghĩa như tên hàm, một biểu thức, một đoạn chương trình bằng một tên, với cách sử dụng này thì chương trình của chúng ta sẽ ngắn gọn và dễ hiểu hơn.

Ví dụ:

#define output printf("stdio.vn");

Trong hàm main tôi thực hiện câu lệnh sau:

output; // printf("stdio.vn");

Những điểm cần chú ý của chỉ thị #define cho cách sử dụng trên:

Khi định nghĩa một biểu thức ta nên đặt nó trong trong cặp dấu ngoặc tròn.

Ví dụ:

#define SUM 5+8

Khi ta gán size = SUM không xảy ra vấn đề gì nhưng khi gán size = 5 \* SUM thì tương đương với size = 5 \* 5+8 chứ không phải là size = 5 \* (5 + 8). Do đó, người ta thường dùng #define SUM (5+8).

Khi định nghĩa đoạn chương trình gồm nhiều câu lệnh thì ta nên đặt trong cặp ngoặc { và }.

Ví dụ:

#define HELLO { printf(“Hello STDIO\n”); printf(“stdio.vn”); }

void main()

{

bool x = true;

if(x) HELLO;

}

Đoạn chương trình trên sẽ cho ra kết quả sau khi x = true:

Hello STDIO

stdio.vn

Khi gán x = false thì không in ra màn hình.

Nhưng khi ta bỏ ngoặc { và } thì đoạn code sẽ như sau:

#define HELLO printf("Hello STDIO\n"); printf("stdio.vn");

Thì ngay cả khi x = false vẫn in ra màn hình:

stdio.vn

###### Chỉ thị #define có đối số

Ngoài cách sử dụng #define như trên, chúng ta còn có thể dùng #define để định nghĩa các macro có đối giống như hàm. Để rõ hơn thì bạn theo dõi ví dụ định nghĩa một macro tính tổng của 2 giá trị.

#define SUM(x,y) (x)+(y)

Khi đó câu lệnh

int z = SUM(x\*2, y\*3);

Được thay bằng

int z = (x\*2) + (y\*3);

Các điểm cần lưu ý:

Giữa macro và dấu ( không được tồn tại khoảng trắng.

Để tránh rủi ro không mong muốn thì khi viết các biểu thức định nghĩa cho macro, các đối tượng hình thức (như x và y ở ví dụ trên) thì nên có cặp ngoặc ( và ) bao quanh. Để minh họa cho điều này thì ta đến với ví dụ sau:

#define MUL(x,y) x\*y

void main()

{

printf("%d",MUL(5+3, 10));

}

Khi đó trình biên dịch thay MUL(5+3, 10) bằng 5+3\*10 và ta nhận đáp án 35 thay vì 80 như ta mong muốn.

##### Chỉ thị #undef

Cú pháp:

#undef identifier

Khi ta cần định nghĩa lại một tên mà ta đã định nghĩa trước đó thì ta sử dụng #undef để hủy bỏ định nghĩa đó và sử dụng #define định nghĩa lại cho tên đó.

Ví dụ:

#define STDIO "Hello STDIO" // Định nghĩa cho tên STDIO là "Hello STDIO"

#undef STDIO // Hủy bỏ định nghĩa cho tên STDIO

#define STDIO "Welcome to STDIO" // Định nghĩa lại cho tên STDIO là "Welcome to STDIO"

#### Chỉ thị biên dịch có điều kiện

Ở nhóm này gồm các chỉ thị #if, #elif, #else, #ifdef, #ifndef.

##### Các chỉ thị #if, #elif, #else.

Cú pháp:

#if constant-expression\_1

// Đoạn chương trình 1

#elif constant-expression\_2

// Đoạn chương trình 2

#else

//Đoạn chương trình 3

#endìf

Nếu constant-expression\_1 true thì chỉ có đoạn chương trình 1 sẽ được biên dịch, trái lại nếu constant-expression\_1 false thì sẽ tiếp tục kiểm ta đến constan-expression\_2. Nếu vẫn chưa đúng thì đoạn chương trình trong chỉ thị #else được biên dịch .

Các constant-expression là biểu thức mà các toán hạng trong đó đều là hằng, các tên đã được định nghĩa bởi các #define cũng được xem là các hằng.

##### Các chỉ thị #ifdef, #ifndef.

Một cách biên dịch có điều kiện khác đó là sử dụng #ifdef và #ifndef, được hiểu như là Nếu đã định nghĩa và Nếu chưa được định nghĩa.

###### Chỉ thị #ifdef.

#ifdef identifier

//Đoạn chương trình 1

#else

//Đoạn chương trình 2

#endif

Nếu indentifier đã được định nghĩa thì đoạn chương trình 1 sẽ được thực hiện. Ngược lại nếu indentifier chưa được định nghĩa thì đoạn chương trình 2 sẽ được thực hiện.

###### Chỉ thị #ifndef

#ifndef identifier

//Đoạn chương trình 1

#else

//Đoạn chương trình 2

#endif

Với chỉ thị #ifndef thì cách thức hoạt động ngược lại với #ifdef.

Ví dụ:

#ifdef MAX // Nếu MAX đã được định nghĩa

#undef MAX // Hủy bỏ MAX

#define MAX 100 // Định nghĩa lại MAX

#else // Nếu MAX chưa được đinh nghĩa

#define MAX 1 // Định nghĩa MAX

#endif

Các chỉ thị điều kiện ở trên, thường được sử dụng cho việc xử lý xung đột thư viện khi chúng ta #include nhiều thư viện như ở ví dụ dưới đây:

Tôi có một file A.h.

//file A.h

Source code A

Giả sử có các file B.h và C.h và 2 file này đều cần nội dung của file A.h, vì thế tôi #include "A.h" vào file B.h và C.h.

//file B.h

#include"A.h"

Source code B

//file C.h

#include"A.h"

Source code C

File main.cpp của tôi #include "B.h" và #include "C.h", khi đó nội dụng file main.cpp trở thành.

//file main.cpp

#include"B.h"

#include"C.h"

Source code file main

Chúng ta có thể hình dung file main.cpp như sau:

//file main.cpp

//#inlude"B.h"

Source code A

Source code B

//#include"C.h"

Source code A

Source code C

Source code file main

Như ta thấy nội dung file A.h sẽ được chép 2 lần sang file main.cpp, bởi vì khi ta #include "B.h" thì nội dung file B.h(có cả nội dung file A.h) đã được chép sang file main.cpp. Ta tiếp tục #include "C.h" thì nội dung file C.h (có cả nội dung file A.h) đã được chép sang file main.cpp. Vì thế, nội dung của file A.h được chép 2 lần trong file main.cpp và khi ta biên dịch thì trình biên dịch sẽ báo lỗi. Để khắc phục lỗi này thì tôi sử dụng chỉ thị #ifndef, #define vào trong file A.h.

#ifndef \_\_A\_H\_\_

#define \_\_A\_H\_\_

Source code A

#endif // \_\_AH\_\_

Khi đó nội dung file main.cpp chúng ta có thể hình dung:

//file main.cpp

//#include"B.h"

#ifndef \_\_A\_H\_\_

#define \_\_A\_H\_\_

Source code A

#endif // \_\_AH\_\_

Source code B

//#include"C.h"

#ifndef \_\_A\_H\_\_

#define \_\_A\_H\_\_

Source code A

#endif // \_\_AH\_\_

Source code C

Source code file main

Cơ chế hoạt động:

Từ dòng 4-9: \_\_A\_H\_\_ chưa được định nghĩa nên cho phép chép nội dung file A.h vào main.cpp.

Dòng 11: Nội dung file B.h.

Từ dòng 14-19: Ở trên \_\_A\_H\_\_ đã được định nghĩa nên không cho phép chép nội dung file A.h vào main.cpp.

Dòng 21: Nội dung file C.h.