

Trong đoạn mã này, bạn sử dụng các toán tử trả về và tham chiếu (\*&) trong `printf("%s\n", *&p)`; . Hãy cùng phân tích chi tiết để hiểu cách hoạt động của mã này.

```
c Sao chép mã  
  
#include<stdio.h>  
  
int main()  
{  
    char *p;  
    p = "hello";  
    printf("%s\n", *&p);  
    return 0;  
}
```

Trong đoạn mã này, bạn sử dụng các toán tử trả về và tham chiếu (\*&) trong `printf("%s\n", *&p)`; . Hãy cùng phân tích chi tiết để hiểu cách hoạt động của mã này.

```
c  
Sao chép mã  
#include<stdio.h>  
  
int main()  
{  
    char *p;  
    p = "hello";  
    printf("%s\n", *&p);  
    return 0;  
}
```

## Phân tích từng phần

### 1. Khai báo và khởi tạo con trỏ p:

- o `char *p;` khai báo p là một con trỏ trỏ đến một chuỗi ký tự.
- o `p = "hello";` gán địa chỉ của chuỗi ký tự hằng "hello" cho con trỏ p. Lúc này, p trỏ đến chuỗi "hello" trong bộ nhớ.
- o Lưu ý rằng chuỗi ký tự "hello" là một chuỗi hằng, nên p trỏ tới một vùng nhớ không thể thay đổi.

### 2. Biểu thức \*&p:

- o Để hiểu \*&p, ta cần phân tích từng toán tử \* và &:
  - `&p`: Lấy địa chỉ của con trỏ p.
  - `*&p`: Toán tử & và \* hủy lẫn nhau, nên `*&p` chỉ đơn giản là p.
  - `*&*p`: Tương tự, `*&*p` cũng chỉ là p vì mỗi cặp \*& sẽ hủy lẫn nhau.
- o Do đó, `*&*p` thực chất chỉ là p.


### 3. `printf("%s\n", *&p);`:

- o Vì `*&*p` là p, câu lệnh `printf("%s\n", *&p);` tương đương với `printf("%s\n", p);`.
- o Trong `printf`, %s sẽ in ra chuỗi ký tự mà p trỏ đến, tức là "hello".

## Kết quả đầu ra

Chương trình sẽ in ra:

```
hello
```

 Sao chép mã

## Tóm tắt

Biểu thức `*&*p` có vẻ phức tạp nhưng thực tế chỉ là `p` do các toán tử `*` và `&` liên tiếp hủy lẫn nhau. Vì vậy, `printf("%s\n", *&*p);` thực sự tương đương với `printf("%s\n", p);`, và chương trình sẽ in ra chuỗi `"hello"`.