Bài 5: Goto - setjmp.h

1. Goto trong C

" goto " là một từ khóa trong ngôn ngữ lập trình C, cho phép chương trình nhảy đến một nhãn (label) đã được đặt trước đó trong cùng một hàm. Mặc dù nó cung cấp khả năng kiểm soát flow của chương trình, nhưng việc sử dụng goto thường được xem là không tốt vì nó có thể làm cho mã nguồn trở nên khó đọc và khó bảo trì.

Ví dụ:

```c
#include <stdio.h>

int main() {
    int i = 0;

    // Đặt nhãn
    start:
        if (i >= 5) {
            goto end;  // Chuyển control đến nhãn "end"
        }

        printf("%d ", i);
        i++;

        goto start;  // Chuyển control đến nhãn "start"

    // Nhãn "end"
    end:
        printf("\n");

    return 0;
}
```

Trong ví dụ này, goto được sử dụng để tạo một vòng lặp đơn giản. Khi i đạt đến giá trị 5, control sẽ chuyển đến nhãn "end" và kết thúc chương trình.

Việc sử dụng goto có thể làm cho mã nguồn trở nên khó bảo trì và khó đọc. Nhiều lập trình viên và chuẩn coding conventions không khuyến khích sử dụng goto nếu có cách thức khác để thực hiện logic. Thay vào đó, sử dụng các cấu trúc kiểm soát vòng lặp (for, while, do-while) và cấu trúc điều kiện (if, else) để viết mã nguồn dễ đọc và bảo trì hơn.

Mặc dù sử dụng goto không được khuyến khích trong phần lớn các trường hợp vì nó có thể làm tăng khả năng phức tạp và khó bảo trì của mã nguồn, nhưng có một số tình huống cụ thể mà việc sử dụng goto có thể được coi là hợp lý:

a. Thoát khỏi nhiều cấp độ vòng lặp

Trong một số trường hợp, việc thoát khỏi nhiều cấp độ vòng lặp có thể trở nên phức tạp nếu sử dụng cấu trúc kiểm soát vòng lặp thông thường. Trong tình huống như vậy, goto có thể được sử dụng để dễ dàng thoát khỏi nhiều cấp độ vòng lặp.

Ví dụ:

```
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
        if (some_condition(i, j)) {
            goto exit_loops;
        }
    }
}

exit_loops:
```

b. Xử lý lỗi và giải phóng bộ nhớ

Trong trường hợp xử lý lỗi, có thể sử dụng goto để dễ dàng giải phóng bộ nhớ đã được cấp phát trước khi thoát khỏi hàm.

```c
void process_data() {
    int *data = malloc(sizeof(int) * 100);
    if (data == NULL) {
        goto cleanup;
    }

    // Xử lý dữ liệu ở đây

    cleanup:
    free(data);
}
```

c. Implement Finite State Machines (FSM)

Trong một số trường hợp, đặc biệt là khi triển khai Finite State Machines, goto có thể được sử dụng để chuyển đến các trạng thái khác nhau một cách dễ dàng.

```c
switch (current_state) {
    case STATE_A:
        // Xử lý State A
        if (condition) {
            goto STATE_B;
        }
        break;

    case STATE_B:
        // Xử lý State B
        break;
}
```

2. Thư viện setjmp.h

setjmp.h là một thư viện trong ngôn ngữ lập trình C, cung cấp hai hàm chính là setjmp và longjmp. Cả hai hàm này thường được sử dụng để thực hiện xử lý ngoại lệ trong C, mặc dù nó không phải là một cách tiêu biểu để xử lý ngoại lệ trong ngôn ngữ này.

Ví dụ:

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf buffer;

void risky_function() {
    printf("Entering risky_function\n");

    // Thiết lập điểm cho việc "quay lại"
    if (setjmp(buffer) != 0) {
        printf("Exiting risky_function due to
longjmp\n");
        return;
    }

    // Mô phỏng một tình huống lỗi
    int error_condition = 1;
    if (error_condition) {
        printf("Error detected in risky_function\n");
        longjmp(buffer, 1);  // "Quay lại" tới điểm đã
được thiết lập bởi setjmp
    }

    printf("Exiting risky_function normally\n");
}

int main() {
```

```
    printf("Starting main\n");
    risky_function();
    printf("Back in main after risky_function\n");

    return 0;
}
```

Trong ví dụ này, setjmp được sử dụng để thiết lập một điểm, và longjmp được sử dụng để "quay lại" điểm đã được thiết lập đó khi một tình huống lỗi được phát hiện trong hàm risky_function.