**Chapter 4. Memory Management Exercises**

**Ex1. Chương trình sau đây có lỗi gì, tại dòng nào, giải thích tại sao**

#include <stdio.h>

#include <stdlib.h>

void print\_values(int \*arr, int size) {

    for (int i = 0; i <= size; i++) {

        printf("%d\n", arr[i]);

    }

}

int main() {

    int \*numbers;

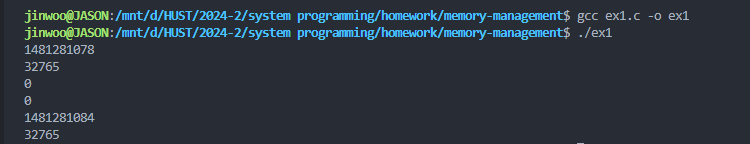
    print\_values(numbers, 5);

    return 0;

}

Chương trình trên có 2 lỗi tại

1. Tại dòng for loop (intex out of bounds). Ví dụ như với các tham số được gán ở hàm main, bởi i <= sive, chương trình trên có thể tham chiếu tới arr[5] dẫn đến bị lỗi.
2. Tại dòng int \*numbers, biến này đã được khai báo nhưng chưa được gán một địa chỉ hợp lệ nào



**Ex2. Chương trình sau có lỗi gì, tại dòng nào, hậu quả ra sao ?**

#include <stdio.h>

void calculate\_average() {

    float scores[5];

    float sum = 0.0;

    for (int i = 0; i < 5; ++i) {

        sum += scores[i];

    }

    printf("Average: %.2f\n", sum / 5);

}

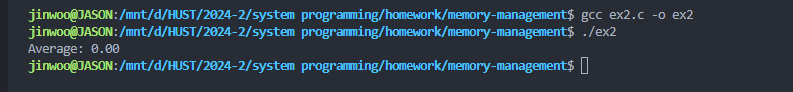
int main() {

    calculate\_average();

    return 0;

}

Chương trình trên có lỗi từ dòng float scores[5], biến này đã được khai báo nhưng chưa được gán bất cứ giá trị nào. Do đó, khi đến vòng lặp, dòng code sum += soces[i], biến sum sẽ được cộng vào các giá trị không xác định. Điều này có thể dẫn đến các kết quả đầu ra không mong muốn, hoặc bị lỗi biên dịch



**Ex3. Chương trình có lỗi gì, ở dòng nào, hậu quả ra sao**  
#include <stdio.h>

#include <string.h>

void copy\_input(const char \*input) {

    char buffer[8];

    strcpy(buffer, input);

    printf("Buffer: %s\n", buffer);

}

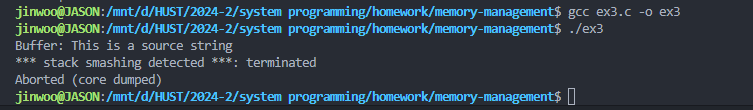
int main() {

    copy\_input("This is a source string”);

    return 0;

}

Đoạn code trên bị lỗi tràn bộ đệm. Tại hàm copy\_input, buffer chỉ được khai báo là một mảng với 8 phần tử (7 kí tự và 1 kí tự /n để kết thúc chuỗi). Tuy nhiên hàm strcpy(buffer, input) lại không kiểm tra kích thước của giá trị đầu vào dẫn đến tràn bộ đệm. Lỗi này khiến chương trình bị crash (segmentation fault)



**Ex4. Chương trình có lỗi gì, ở dòng nào, hậu quả ?**

#include <stdio.h>

int\* create\_value() {

    int value = 100;

    return &value;

}

int main() {

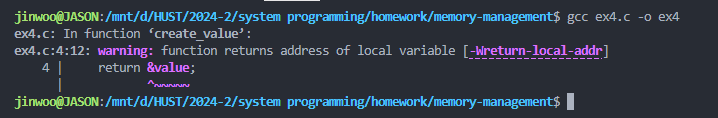
    int \*ptr = create\_value();

    printf("Value: %d\n", \*ptr);

    return 0;

}

Tại chương trình trên, hàm create\_value() trả về địa chỉ của một biến cục bộ value. Sau khi chương trình được thực thi xong thì không gian vùng nhớ của biến value sẽ bị giải phóng dễn đến địa chỉ &value trở đến một địa chỉ không hợp lệ. Điều này gây ra lỗi biên dịch chương trình



**Ex5. Chương trình có lỗi gì, hậu quả ?**

#include <stdlib.h>

void process() {

    int \*data = malloc(sizeof(int) \* 10);

    free(data);

    // Some unrelated code...

    if (data != NULL) {

        free(data);

    }

}

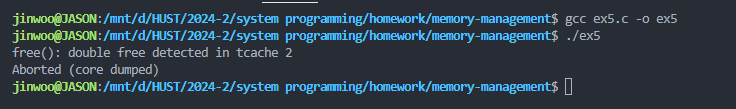
int main() {

    process();

    return 0;

}

Chương trình trên bị lỗi doube free (giải phóng bộ nhớ 2 lần). Trong chương trình trên, sau khi free(data), giá trị của biến data không phải là NULL mà vẫn là địa chỉ mà nó đã được cấp phát tuy nhiên địa chỉ này không được khả dụng để truy nhập. Do đó khi kiểm tra data với NULL, chương trình sẽ giải phóng biến data hai lần.



**Ex6. Chương trình có lỗi gì, hậu quả ?**

#include <stdio.h>

#include <stdlib.h>

int\* allocate\_and\_free() {

    int \*ptr = malloc(sizeof(int));

    \*ptr = 123;

    free(ptr);

    return ptr;

}

int main() {

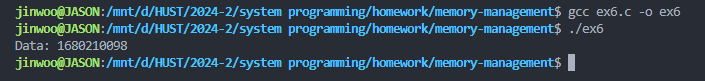
    int \*data = allocate\_and\_free();

    printf("Data: %d\n", \*data);

    return 0;

}

Chương trình trên bị lỗi use after free. Sau khi ptr đã được free tại hàm allocate\_and\_free() thì lại được trả về ở cuối hàm này, sau đó lại được sử dụng sau đó ở hàm main(). Điều này có thể dẫn đến các kết quả đầu ra không mong muốn. Ví dụ ở đây, hàm printf đã in ra một giá trị rác mà chương trình đã ghi đề vào sau khi free



**Ex7. Chương trình có thể gây vấn đề gì ? Lỗi nằm ở đâu ?**

#include <stdio.h>

#include <stdlib.h>

void load\_data() {

    char \*buffer = malloc(256);

    fgets(buffer, 256, stdin);

    printf("Input: %s\n", buffer);

}

int main() {

    for (int i = 0; i < 1000; ++i) {

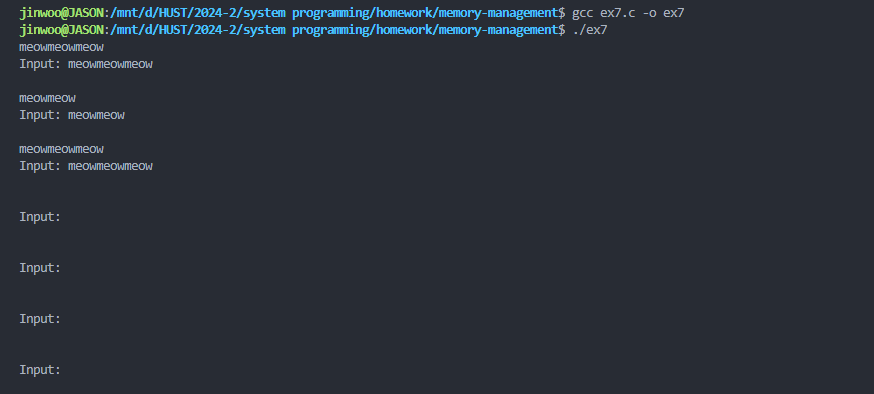
        load\_data();

    }

    return 0;

}

Chương trình trên có lỗi gây ra rò rỉ bộ nhớ (memory leak). Các bộ nhỡ đã được cấp phát tại hàm load\_data() tuy nhiên lại không được free sau đó. Điều này lặp lại 1000 lần dẫn đến khối lượng lớn bộ nhớ được cấp phát sử dụng nhưng không được sử dụng dẫn đến lãng phí tài nguyên hệ thống



**Ex8. Chương trình sau có lỗi ở đâu, hãy sửa lại**

#include <stdio.h>

#include <stdlib.h>

int main() {

    int \*\*p;

    int N = 10, M = 10;

    printf("size of int: %d, size of int\*: %d\n", sizeof(int), sizeof(int\*));

    p = malloc(N \* sizeof(int));

  for (int i = 0; i < N; i++) {

        p[i] = malloc(M \* sizeof(int));

    }

    return 0;

}

Trong chương trình trên, p là con trỏ tới contror, nên phải được cấp phát với kích thước của con trỏ chứ không phải cấp phát với kích thước của int như ở dòng trên.

Sửa lại

#include <stdio.h>

#include <stdlib.h>

int main() {

    int \*\*p;

    int N = 10, M = 10;

    printf("size of int: %d, size of int\*: %d\n", sizeof(int), sizeof(int\*));

    p = malloc(N \* sizeof(int \*));

  for (int i = 0; i < N; i++) {

        p[i] = malloc(M \* sizeof(int));

    }

    return 0;

}

**Ex9. Chương trình sau có lỗi ở đâu, hãy sửa lại**

int \*BinheapDelete(int \*\*binheap, int \*size) {

    int \*packet;

    packet = binheap[0];

    binheap[0] = binheap[\*size - 1];

    \*size--;

    Heapify(binheap, \*size, 0);

    return(packet);

}

Đoạn code trên có cú pháp không hợp lệ tại dòng \*size--, bị sai logic khi muốn giảm contror size

Sửa lại

int \*BinheapDelete(int \*\*binheap, int \*size) {

    int \*packet;

    packet = binheap[0];

    binheap[0] = binheap[\*size - 1];

    (\*size)--;

    Heapify(binheap, \*size, 0);

    return(packet);

}

**Ex10. Cho biết kết quả in ra màn hình**

#include <stdio.h>

int main() {

    char \*str = "Programming";

    char \*ptr = str;

    printf("%c\n", \*ptr + 1);

    printf("%c\n", \*(ptr + 1));

  return 0;

}

