Easy lab 实验报告

实验内容

2. 需求：
   - 本次实验需要大家在注释掉 `cout<<"A magic print! If you comment this, the program will break."<<endl;` 后，修复这个段错误的bug；
   - 然后提交一个文档，具体说明你怎么找到的这个修复方法，并且阐述为什么会出现这个问题，最好能写一段代码从内存布局上基于证明。

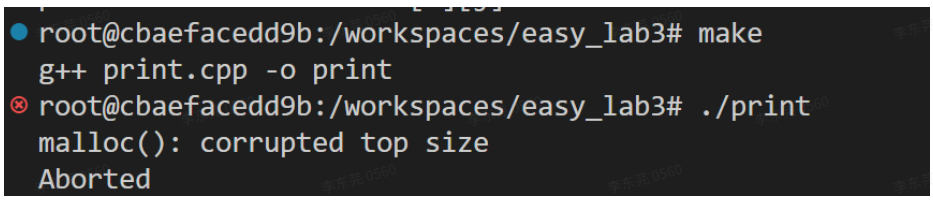
我的实现

本次lab的主要实现代码在 `print.cpp` 中。

当我注释掉这行代码：

```
1 cout<<"A magic print! If you comment this, the program will break."<<endl;
```

并运行 `make` 和 `./print` 指令后，系统提示内存异常（即发生segmentaiton fault）



内存异常运行截图

在我对代码的深入分析中，我注意到了一个关键的内存管理问题。

我发现，在 `int **double_array(size_t n)` 函数中，原本的代码使用 `new int*[8]` 分配了一个指针数组，这意味着无论 `n` 的大小如何，程序总是只为 `result` 分配了8个指针。实际上，程序需要根据 `n` 的值动态分配内存。这就是为什么当 `n` 大于8时（在提供的代码为n = 64），程序会尝试访问未被分配的内存区域，从而引发越界错误。

进一步地，我尝试使用AddressSanitizer对于具体的内存问题进行定位：

```
root@cbaefacedd9b:/workspaces/easy_lab3# g++ -fsanitize=address -g print.cpp -o main
root@cbaefacedd9b:/workspaces/easy_lab3# ./main
=================================================================
==5268==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x606000000060 at pc 0x55c1981303df bp 0x7ffe8f395830 sp 0x7ffe8f395820
WRITE of size 8 at 0x606000000060 thread T0
    #0 0x55c1981303de in double_array(unsigned long) /workspaces/easy_lab3/print.cpp:14
    #1 0x55c1981304af in main /workspaces/easy_lab3/print.cpp:25
    #2 0x7f06855d3d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0x7f06855d3e3f in __libc_start_main_impl ../csu/libc-start.c:392
    #4 0x55c1981302a4 in _start (/workspaces/easy_lab3/main+0x12a4)

0x606000000060 is located 0 bytes to the right of 64-byte region [0x606000000020,0x606000000060)
allocated by thread T0 here:
    #0 0x7f0685bbb357 in operator new[](unsigned long) ../../../../src/libsanitizer/asan/asan_new_delete.cpp:102
    #1 0x55c198130382 in double_array(unsigned long) /workspaces/easy_lab3/print.cpp:11
    #2 0x55c1981304af in main /workspaces/easy_lab3/print.cpp:25
    #3 0x7f06855d3d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

SUMMARY: AddressSanitizer: heap-buffer-overflow /workspaces/easy_lab3/print.cpp:14 in double_array(unsigned long)
```

AddressSanitizer运行截图

通过这条输出，我们可以确认问题出现在 `int **double_array(size_t n)` 函数中，产生了 `heap-buffer-overflow` 异常。

为了解决这个问题，我将内存分配语句 `new int*[8]` 改为 `new int*[n]`，这样就可以根据实际需求动态分配内存。

修改后的代码如下所示：

```
int **double_array(size_t n) {
    int **result = new int*[n];
        for (int i = 0; i < n; ++i) {
            result[i] = matrix[i];
            for (int j = 0; j < n; ++j){
                result[i][j] = j;
            }
    }
    return result;
}
```

在这个修正后的版本中，我为 `result` 分配了足够数量的指针，并且每个指针指向 `matrix` 数组的一个行数组，即 `result[i]` 和 `matrix[i]` 指向相同的内存地址（共享内存）。

这种内存共享策略也意味着在释放 `result` 时不能释放它指向的每一行内存，因为这些内存实际上是属于 `matrix` 的，会在程序的其他部分中继续使用。

因此，我只需要释放为 `result` 指针本身分配的内存。因为我是用 `new[]` 进行内存分配的，需要使用 `delete[]` 而不是 `free()` 以下是修改后的代码：

```
delete[] result;  // 释放result指针数组的内存
```

再次运行 `make` 和 `./print` 指令后，程序正常输出，无报错。