**内存泄漏检查器**

背景（最好不要跳过这部分，这里有一些有用的信息。） 让C程序具有内存安全性非常困难！你需要确保每个内存访问都是有效的，并且所有动态分配的内存都被正确释放（free）。有没有一种工具可以自动检查内存管理，以便可以在不进行手动调试的情况下检测到与内存相关的错误？ 其实我们已经有一些非常有效的工具来检测这些烦人的错误。GCC 和 Clang 都有 AddressSanitizer（论文）和 UndefinedBehaviorSanitizer。Clang 还有 MemorySanitizer。Valgrind 是一个著名的用于检测内存管理和线程错误的工具集，但它只支持 Linux 和 Mac OS X。但是 Windows 用户不必感到沮丧，因为世界上最好的集成开发环境 Visual Studio 已经拥有所有这些工具，可以为你处理一切。

**描述**

在这个问题中，你将编写一个非常简单的工具来检测内存泄漏和无效的 free 调用。内存泄漏是指堆上分配了一些内存块（通过 malloc、calloc、realloc 或 aligned\_alloc），但没有释放（free）。当且仅当满足以下条件时，带有参数 ptr 的 free 调用才有效： ptr 是空指针，或者 ptr 是由先前对 malloc、calloc、realloc 或 aligned\_alloc 的调用返回的指针，且这是第一次将其传递给 free。

在开始前，请确保你已经阅读了有关 malloc、calloc 和 free 的 cppreference 页面。你可以忽略关于线程安全和“同步”方面的段落。不要相信来自其他来源的资料，如维基百科、百度知道、CSDN、知乎等。

**主要思路**

检测内存泄漏的思路很简单： 首先，我们需要记录每次对 malloc 和 calloc 的调用。我们可以使用一些数据结构（例如数组）来存储分配的内存的地址。 由于你们中的大多数人对其他花哨的数据结构（如哈希表）了解不多，所以使用数组足够完成这个作业。 为简化任务，你不需要关心 realloc 或 aligned\_alloc。 当调用 free(ptr) 时，检查数组中是否存在 ptr。 如果数组包含此地址，则此操作有效。可以安全地释放内存，并从数组中删除相应的记录。（注意：要从数组中删除记录，我们建议您简单地将记录的指针设置为 NULL。） 如果没有，则此次对 free 的调用无效，您应该打印一些错误信息。 当程序终止时

**获取附加信息**

如果不仅可以记录分配的内存地址，还可以记录其他的内存地址，这将非常有用

其他信息，如源文件的名称和行号。例如，你想要看到类似这样的错误消息

100 bytes memory not freed (allocated in file hw3\_problem1.c, line 32)

Invalid free in file hw3\_problem1.c, line 49

#define MAX\_RECORDS 1000

struct Record {

void \*ptr; // address of the allocated memory

size\_t size; // size of the allocated memory

int line\_no; // line number, at which a call to malloc or calloc happens

const char \*file\_name; // name of the file, in which the call to malloc or calloc happens

};

struct Record records[MAX\_RECORDS];

void fun(void) \_\_attribute\_\_((destructor));

void fun(void) {

printf("Goodbye world!\n");

}

int main(void) {}

**详细的需求**

关于如何处理某些情况以及打印什么内容的详细要求如下。

free(NULL)应该什么都不做。

如果malloc或calloc返回NULL(表示失败)，则不会分配内存，也不会记录任何内容。

当发生无效的释放时，您应该(在函数recorded\_free中)

检测并输出

Invalid free in file FILENAME, line N

以换行符('\n')结尾，其中FILENAME被替换为文件名，n被替换为行号，以及

防止调用真正的空闲函数，这样实际上就不会发生无效的空闲。

在程序结束时(即在函数leak\_check中)，大家应该总结该程序的内存使用情况:

首先,输出

Summary:

末尾有一个换行符。

遍历数组，查看是否有内存是空闲的d。对于每一条这样的记录，

输出

M bytes memory not freed (allocated in file FILENAME, line N)

在末尾有一个换行符，其中M被替换为该内存块的大小(以字节为单位)，FILENAME被替换为文件名，N被替换为行号。这些行可以按任何顺序打印。你不需要关心byte的单数/复数形式。即使M == 1，也只打印字节。

作为结论，如果检测到内存泄漏，则输出

N allocation records not freed (M bytes in total).

在末尾有一个换行符，其中N被替换为非空闲的分配记录的数量d, M被替换为泄漏内存的总大小(以字节为单位)。

你不需要关心“record”的单数/复数形式。即使N == 1，也只打印记录。

如果没有内存泄漏，则输出

All allocations have been freed.

末尾有一个换行符。

**你应该知道一些特别的事情**

malloc(0)、calloc(0, N)和calloc(N, 0)的行为是实现定义的。

它们可能返回一个null指针(没有分配内存)，但它们也可能分配一些内存并返回一个指向内存的指针。只要返回的指针是非空的，就应该记录下来，如果该内存不是空闲的，也会被认为是内存泄漏。

由于对齐要求或其他可能的原因，malloc和calloc分配的内存可能比预期的要多。例如，前面提到过，calloc(N, M)可能会分配超过N \* M字节的内存，而malloc(0)可能会分配一些内存。

但你不需要考虑这些复杂的事情——只需要对malloc(N)计数N，对calloc(N, M)计数N \* M。换句话说，下面是一个可能的汇总输出:

Summary:

0 bytes memory not freed (allocated in file a.c, line 10)

1 allocation records not freed (0 bytes in total).

**提示**

我们为您提供了一个模板代码memcheck.c，其中包含上面提到的所有内容。

这保证了调用malloc和calloc的总次数不会超过MAX\_RECORDS，即1000。

如果需要，可以随意添加其他辅助函数。如果您遇到名称冲突问题，请联系我们。

您可以尝试一些花哨的数据结构，如链表或散列表，以优化时间和空间复杂度。但这些不会给你额外的分数。

**提交**

将memcheck.h提交给OJ(或复制粘贴其内容)。

**进一步**

到目前为止还不错。你已经写了一些你可以真正利用的东西。虽然它的功能有限，但可以使用相同或类似的思想很容易地扩展它。例如:

添加对realloc和aligned\_alloc的支持，可能还支持C23的free\_sized和

free\_aligned\_sized。

在无效的空闲呼叫中检测双空闲。“Double- free”通常指的是一个调用自由(ptr)，其中ptr指向的内存已经是空闲的d。

此外，我们用自己的函数替换malloc, calloc和free的方式是通过预处理器指令#define，这个指令有点弱，可能会导致问题。例如,

我们要检查的每个源代码文件都必须包含"memcheck.h"，这可能会导致在链接时重新定义符号。用户代码也可以通过#undef这些宏来避免这种情况。

一个可能更好的解决方案是破解这些函数的动态链接，例如使用GNU C库提供的一些魔法，或malloc挂钩。自己搜索网络。

c++中有一些语言结构允许用户声明替代内存分配和释放函数(new运算符和delete运算符)，这样你就不必这么做了

用“hack”的方式来做。详情请参阅我的视频。

**testcases:**

和OJ的测试用例几乎一样。(但他们也有不同之处。不要硬编码结果。)你可以运行`.c`文件，看看它们的输出是否与对应的`.c`文件相同。出的文件。

测试用例`fhq\_treap`有一个输入文件`fhq\_treap`。`，并且`fhq\_treap.c`的`stdin`已经使用`freopen`函数重定向到它。`fhq\_treap.c`是一段可以传递Luogu P3369的代码。

所有的`。Out ' and '。in`文件是纯测试文件，可以使用任何代码编辑器(当然包括VSCode)打开。

`.c`文件使用的是`#include "../memcheck.h"`而不是`#include "memcheck.h"`，因为`memcheck.h`位于`testcases` (`..'表示父目录)。