CS 213，2001 年秋季

Malloc 实验室：编写动态存储分配器 已分配：11 月 2 日星期五，到期：11 月 20 日星期二 11:59PM

Cory Williams（cgw@andrew.cmu.edu) 是这项任务的牵头人。

1 导言

在本实验中，您将为 C 程序编写一个动态存储分配器，即您自己版本的 `malloc`、`free` 和 `realloc` 例程。我们鼓励你创造性地探索设计空间，并实现一个正确、高效和快速的分配器。

2 物流

您最多可以两人一组。对作业的任何说明和修改都将公布在课程网页上。

3 分发说明

针对站点：在此插入一段文字，说明学生应如何下载 `malloclab-handout.tar` 文件。

首先，将 `malloclab-handout.tar` 复制到一个受保护的目录中，并计划在该目录中进行工作。然后执行命令：`tar xvf malloclab-handout.tar`，这将导致许多文件解压到该目录中。

`mdriver.c` 程序是一个驱动程序，可用于评估解决方案的性能。使用 `make` 命令生成驱动程序代码，并使用 `./mdriver -V` 命令运行它。(`V` 标志将显示有用的摘要信息）。

查看文件 `mm.c`，你会发现一个 C 结构团队，你应该在其中插入所需的关于组成编程团队的一两个人的识别信息。请**立即执行，以免遗忘。**

完成实验后，您只需上交一个包含解决方案的文件 (`mm.c`)。

4 如何在实验室工作

动态存储分配器将由以下四个函数组成，这些函数在 mm.h 中声明
并在 mm.c 中定义

```
int   mm_init(void);
void *mm_malloc(size_t size);
void mm_free(void *ptr);
void *mm_realloc(void *ptr, size_t size);
```

我们给你的 mm.c 文件实现了我们能想到的最简单但功能上仍然正确的 malloc 包。以此为起点，修改这些函数（也可能定义其他私有静态函数），使其符合以下语义：

- mm_init：在调用 mm malloc mm realloc 或 mm free 之前，应用程序（即你将用来评估实现的跟踪驱动程序）会调用 mm init 进行必要的初始化，例如分配初始堆区域。如果初始化过程中出现问题，返回值应为-1，否则为 0。

- 毫米 malloc：mm malloc 例程会返回一个指向已分配块有效负载的指针，其大小至少为字节。整个已分配块应位于堆区域内，不应与任何其他已分配块重叠。

  我们将把您的实现与标准 C 库（libc）中提供的 malloc 版本进行比较。由于 libc malloc 总是返回 8 字节对齐的有效载荷指针，因此您的 malloc 实现也应如此，总是返回 8 字节对齐的指针。

- mm_free：mm free 例程释放 ptr 指向的数据块。它不返回任何内容。只有当传递的指针（ptr）是在调用 mm malloc 或 mm realloc 之前返回的，且尚未被释放时，该例程才能保证有效。

- mm_realloc：mm realloc 例程会返回一个指向已分配区域的指针，其大小至少为字节，并有以下限制条件。

  - 如果 ptr 为 NULL，则调用等同于 mm malloc(size)；
  - 如果 size 等于零，则调用等同于 mm free(ptr)；
  - 如果 ptr 不是 NULL，那么它一定是之前调用 mm malloc 或 mm realloc 时返回的

。调用 `mm realloc` 会将 `ptr`（*旧块*）指向的内存块的大小改为 `size` 字节，并返回新块的地址。请注意，新内存块的地址可能与旧内存块的地址相同，也可能不同，这取决于你的实现、旧内存块的内部碎片数量以及 `realloc` 请求的大小。

新代码块的内容与旧 `ptr` 代码块的内容相同，但以新旧大小的最小值为限。其他内容均未初始化。例如，如果旧块为 8 字节，新块为 12 字节，那么新块的前 8 个字节与旧块的前 8 个字节相同。

字节，最后 4 个字节未初始化。同样，如果旧数据块为 8 字节，新数据块为 4 字节，则新数据块的内容与旧数据块的前 4 字节相同。

这些语义与 `libc` 对应的 `malloc`、`realloc` 和 `free` 路由器的语义一致。在 shell 中键入 `man malloc` 以获取完整文档。

5 堆一致性检查器

动态内存分配器是出了名的难以正确高效编程的怪兽。之所以难以正确编程，是因为动态内存分配器涉及大量非类型指针操作。你会发现编写一个堆检查器非常有用，它可以扫描堆并检查堆的一致性。

堆检查程序可能检查的一些示例包括

- 是否免费列表中的每个区块都标记为免费？
- 是否有连续的自由区块以某种方式逃脱了凝聚？
- 是否每个免费区块都在免费列表中？
- 空闲列表中的指针是否指向有效的空闲块？
- 分配的区块是否有重叠？
- 堆块中的指针是否指向有效的堆地址？

你的堆检查器将由 `mm.c` 中的函数 `int mm check(void)` 组成。它将检查任何你认为必要的不变量或一致性条件。如果堆是一致的，它将返回一个非零值。您不必局限于所列建议，也不必检查所有建议。我们鼓励你在检查失败时打印出错误信息。

该一致性检查程序用于开发过程中的调试。在提交 `mm.c` 时，请确保删除对 `mm check` 的任何调用，因为它们会降低吞吐量。您的 `mm check` 函数将获得样式点。请务必在注释中说明并记录您正在检查的内容。

6 支持例程

memlib.c 软件包为动态内存分配器模拟内存系统。您可以在 `memlib.c` 中调用以下函数：

- `void *mem_sbrk(int incr)`：将堆扩展 `incr` 字节，其中 `incr` 为正非零整数，并返回指向新分配堆区域第一个字节的通用指针。除了 `mem sbrk` 只接受一个非零正整数参数外，其语义与 Unix `sbrk` 函数相同。

- `void *mem_heap_lo(void)`：返回堆中第一个字节的通用指针。
- `void *mem_heap_hi(void)`：返回堆中最后一个字节的通用指针。
- `size_t mem_heapsize(void)`：以字节为单位返回堆的当前大小。
- `size_t mem_pagesize(void)`：以字节为单位返回系统页面大小（Linux 系统为 4K）。

7 追踪驱动的驾驶员计划

`malloclab-handout.tar` 分发包中的驱动程序 `mdriver.c` 会测试 `mm.c` 包的正确性、空间利用率和吞吐量。驱动程序由 `malloclab-handout.tar` 发行版中的一组*跟踪文件*控制。每个跟踪文件都包含一系列分配、重新分配和释放指令，指示驱动程序按一定顺序调用 `mm_malloc`、`mm_realloc` 和 `mm_free` 例程。驱动程序和跟踪文件与我们在评定你提交的 `mm.c` 文件时将使用的驱动程序和跟踪文件相同。

驱动程序 `mdriver.c` 接受以下命令行参数：

- `-t <tracedir>`：在 `tracedir` 目录中查找默认跟踪文件，而不是 `config.h` 中定义的默认目录。
- `-f <tracefile>`：使用一个特定的跟踪文件进行测试，而不是使用默认的跟踪文件集。
- `-h`:打印命令行参数摘要
- `-l`:除学生的 malloc 软件包外，运行并测量 `libc` malloc。
- `-v`:详细输出。以紧凑的表格打印每个跟踪文件的性能明细。
- `-V`:更详细的输出。在处理每个跟踪文件时打印额外的诊断信息。在调试过程中，可用于确定是哪个跟踪文件导致了 malloc 程序包的失败。

8 编程规则

- 您不应更改 `mm.c` 中的任何接口。

- 不得调用任何与内存管理相关的库调用或系统调用。这不包括在代码中使用 `malloc`、`calloc`、`free`、`realloc`、`sbrk`、`brk` 或这些调用的任何变体。
- 您不能在 `mm.c` 程序中定义任何全局或静态复合数据结构，如数组、结构体、树或列表。不过，您可以在 `mm.c` 中声明全局标量变量，如整数、浮点数和指针。

- `libc malloc` 软件包返回的块以 8 字节为边界对齐，为了与 `libc malloc` 保持一致，分配器必须始终返回以 8 字节为边界对齐的指针。驱动程序将强制执行这一要求。

9 评估

如果您违反了任何规则，或者您的代码存在漏洞并导致驱动程序崩溃，您将得到**零分**。否则，你的成绩将按以下方式计算：

- *正确性（20 分）*。如果您的解决方案通过了驱动程序的正确性测试，您将获得满分。每条正确的轨迹都将获得部分分数。
- *性能（35 分）*。将使用两个性能指标来评估您的解决方案：
  - *空间利用率*：驱动程序使用的内存总量（即通过 `mm malloc` 或 `mm realloc` 分配但尚未通过 `mm free` 释放的内存）与分配器使用的堆大小之间的峰值比率。最佳比率等于 1。为了使这一比率尽可能接近最佳值，你应该找到好的策略来尽量减少碎片。
  - *吞吐量*：每秒完成的平均操作数。

  驱动程序通过计算*性能指数*来总结分配器的性能、
  P 是空间利用率和吞吐量的加权和

$$P = wU + (1 - w)\min\left(1, \frac{T}{T_{libc}}\right)$$

  其中 U 是空间利用率，T 是吞吐量，而 T_{libc} 则是您的系统在默认跟踪条件下 `libc` malloc 的估计吞吐量[1]性能指数偏向于空间利用率而非吞吐量，默认值为 $w = 0.6$。

  考虑到内存和 CPU 周期都是昂贵的系统资源，我们采用这一公式来鼓励内存利用率和吞吐量的均衡优化。理想情况下，性能指数将达到 $P = w + (1 - w) = 1$ 或 100%。由于每个指标对性能指数的贡献最多分别为 w 和 $1-w$，因此不应走极端，只优化内存利用率或吞吐量。要获得好成绩，必须在利用率和吞吐量之间取得平衡。

- 风格（10 分）。

– 您的代码应分解为多个函数，并尽可能少地使用全局变量。

– 您的代码应该以头注释开始，头注释应描述空闲块和已分配块的结构、空闲列表的组织以及分配器如何操作空闲列表。

[1] T_{libc} 的值是驱动程序中的常数（600 Kops/s），由指导教师在配置程序时设定。

– 每个子程序都应有一个标题注释，说明它的作用和方法。

– 堆一致性检查程序的检查应该彻底，并有详尽的记录。

如果堆一致性检查器良好，您将获得 5 分；如果程序结构和注释良好，您将获得 5 分。

10 交接说明

针对站点：在此插入一段文字，说明学生应如何上交解决方案 `mm.c` 文件。

11 提示

- *使用* `mdriver -f` *选项。*在初始开发期间，使用微小的跟踪文件可以简化调试和测试。我们提供了两个这样的跟踪文件（`short1,2-bal.rep`），供您用于初始调试。
- *使用* `mdriver -v` *和* `-V` *选项。* `-v` 选项将提供每个跟踪文件的详细摘要。此外，`-V` 还会显示每个跟踪文件的读取时间，这将帮助你隔离错误。
- *使用* `gcc -g` *进行编译，并使用调试器。*调试器将帮助你隔离和识别越界内存引用。
- *理解教科书中 malloc 实现的每一行。*教科书中有一个基于隐式空闲列表的简单分配器的详细示例。请以此为出发点。在了解简单隐式列表分配器的所有内容之前，不要开始编写分配器。
- *用 C 预处理器宏封装指针运算。*在内存管理程序中进行指针运算时，由于需要进行各种转换，因此既混乱又容易出错。为指针运算编写宏可以大大降低复杂性。请参阅文中的示例。
- *分阶段执行。*前 9 个跟踪包含对 `malloc` 和 `free` 的请求。最后两个跟踪记录包含对 `realloc`、`malloc` 和 `free` 的请求。我们建议您首先在前 9 个跟踪中正确、高效地运行 `malloc` 和 `free` 例程。然后才将注意力转向 `realloc` 的实现。首先，在现有的 `malloc` 和 `free` 实现基础上构建 `realloc`。但要获得真正的高性能，你需要构建一个独立的 `realloc`。

- *使用剖析器。*您可能会发现 `gprof` 工具有助于优化性能。
- *尽早开始！*只需几页代码就能编写出高效的 malloc 程序包。不过，我们可以保证，这将是你职业生涯中迄今为止编写过的最困难、最复杂的代码。因此，请尽早开始，祝你好运！