在这次lab中我们要写mm.c文件以求完成：malloc、free、realloc的实现，其中有几个函数等待我们填写（包括但不仅限于：堆的初始化、内存分配、内存释放、堆的拓展、空闲块合并）

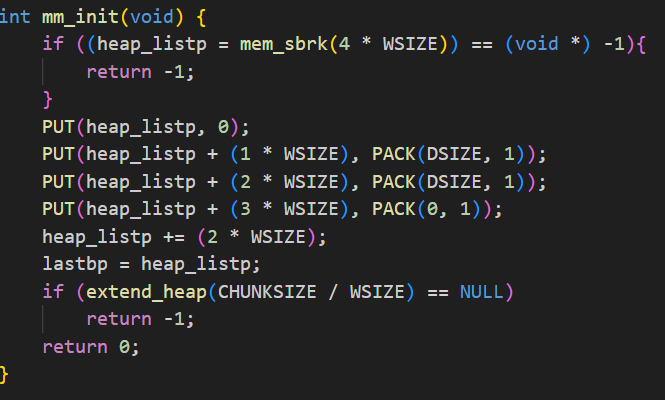
在代码中我们用了一组宏定义，这些宏定义简化了对接下来的内存块的操作，同时还能确保我们内存分配的对齐和正确性



ALIGNMENT 8：定义了内存对齐的字节数为 8 字节，用于保证内存地址按照一定规则对齐，方便内存管理和硬件访问优化等。ALIGN(p) (((size\_t)(p) + (ALIGNMENT - 1)) & ~0x7)：将给定指针p的地址按照ALIGNMENT字节对齐，通过位运算来实现地址的向上对齐操作。SIZEMASK (~0x7)：用于屏蔽内存块大小字段中的低 3 位，获取实际的内存块大小（因为低 3 位可能用于其他标志位等，此处约定按此方式获取大小）。PACK(size, alloc) ((size)|(alloc))：将内存块的大小和分配标志位合并为一个值，方便存储在内存块头部等位置，通常低 1 位用于表示是否已分配（1 表示已分配，0 表示空闲）。getSize(x) ((x)->size & SIZEMASK)：从内存块头部结构体（假设存在这样的结构体，代码中未明确展示完整定义）中提取出内存块的实际大小。还有如WSIZE（表示字大小，通常为 4 字节）、DSIZE（双字大小，8 字节）、CHUNKSIZE（初始堆扩展大小等相关的常量定义）等，用于后续内存操作中涉及的字节数量计算等

然后是mm\_init函数：首先尝试通过mem\_sbrk函数（推测是自定义的用于扩展内存空间的函数，依赖memlib.h）申请 4 个WSIZE大小的内存空间作为初始的堆空间布局。如果申请失败则返回-1表示初始化失败。

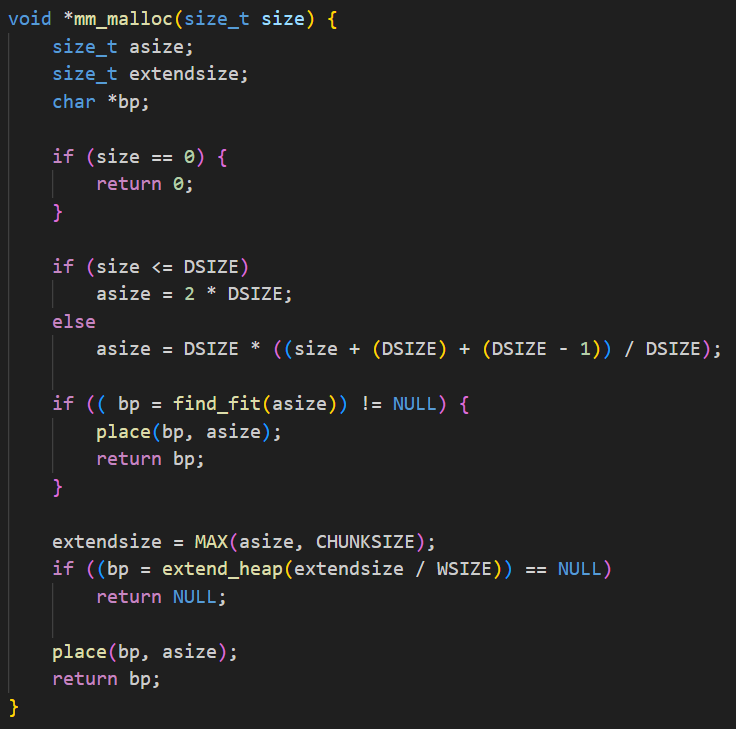
接着初始化堆空间头部的一些信息，设置特定位置的内存块大小和分配标志等，例如将开头部分标记为特殊的占位信息，中间部分设置为初始的已分配的双字大小内存块（可能用于边界等管理），最后部分设置为结束标志等。调整heap\_listp指针指向实际管理的堆起始位置，并记录初始的lastbp（推测用于后续查找空闲块等操作的相关记录）。最后通过extend\_heap函数扩展堆空间到指定的CHUNKSIZE大小，如果扩展失败则返回-1，成功则返回0表示初始化成功。



其次是mm\_malloc函数：首先对传入的请求分配的内存大小size进行处理，如果size为 0 则直接返回0表示空指针分配情况。如果size小于等于DSIZE，则将实际分配大小asize设置为2 \* DSIZE，进行内部对齐和最小分配大小的处理；否则按照一定的对齐和计算规则确定合适的分配大小asize。

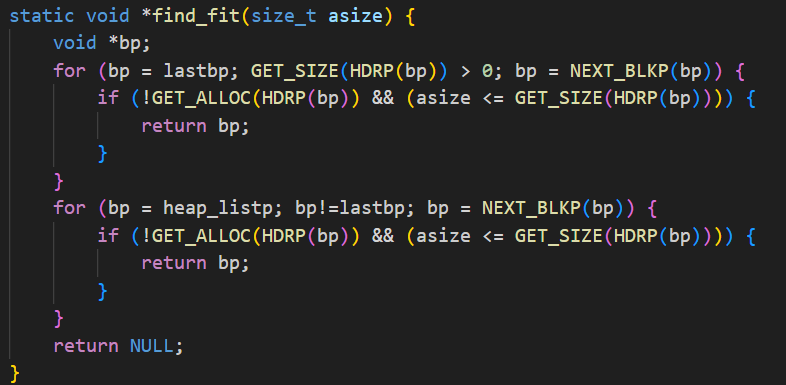
接着调用find\_fit函数尝试查找合适的空闲内存块来满足分配需求，如果找到合适的空闲块（bp不为NULL），则调用place函数进行内存块的分割和分配标记等操作，然后返回分配的内存块指针bp。

如果没有找到合适的空闲块，则计算需要扩展的堆大小（取asize和CHUNKSIZE中的较大值），通过extend\_heap函数扩展堆空间，再调用place函数进行内存分配相关操作，最后返回分配的内存块指针或者NULL（如果扩展失败等情况）。

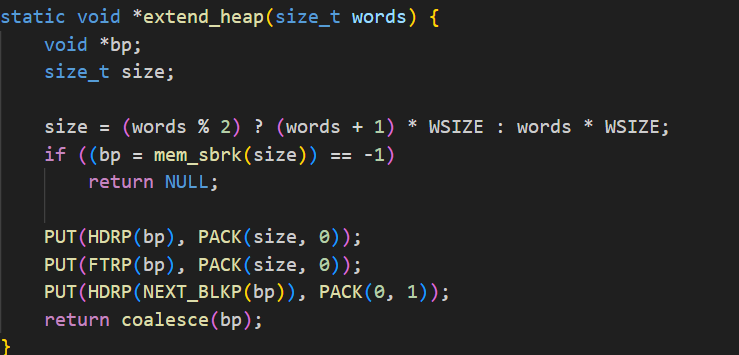


find\_fit函数：实现了两种查找空闲内存块的策略，首先从lastbp开始向后遍历内存块（通过NEXT\_BLKP宏来获取下一个内存块指针），检查每个内存块的头部信息，判断是否为空闲（GET\_ALLOC宏获取分配标志为 0）且大小是否满足请求分配的大小asize，如果满足则返回该内存块指针bp。

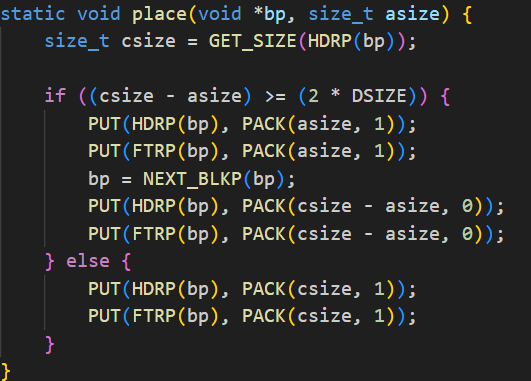
如果第一种遍历没有找到合适的空闲块，则从heap\_listp开始向前遍历直到lastbp（同样通过NEXT\_BLKP宏来移动指针），重复上述空闲块查找判断逻辑，若找到则返回对应指针，若都没找到则返回NULL。



extend\_heap函数：根据传入的words（以字为单位的数量）来计算实际需要扩展的内存大小size，确保其按照WSIZE字节对齐，通过mem\_sbrk函数来扩展堆内存空间，如果扩展失败（返回-1）则返回NULL表示扩展失败。对于新扩展的内存块，设置其头部（HDRP宏获取头部指针）和尾部（FTRP宏获取尾部指针）的内存块大小和分配标志（标记为空闲，即0表示未分配），并设置下一个内存块（新扩展的末尾的下一个位置）的头部信息为结束标志（大小为 0，已分配标志为 1），最后调用coalesce函数来尝试合并新扩展的内存块与相邻的空闲内存块（如果有的话），并返回合并后的内存块指针。

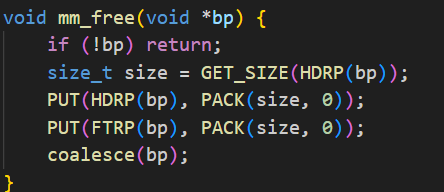


place函数：首先获取当前内存块（指针为bp）的总大小csize，然后判断如果剩余空间（csize - asize）大于等于2 \* DSIZE，意味着有足够空间可以进行分割。此时将当前内存块的头部和尾部分别标记为已分配且大小为asize，然后移动指针bp到下一个内存块位置，将其标记为空闲且大小为剩余的空间大小（csize - asize）。如果剩余空间不足分割，则直接将整个当前内存块标记为已分配且大小保持为csize。



mm\_free函数：首先判断传入的要释放的内存块指针bp是否为空，如果为空则直接返回，不做任何操作。

接着获取要释放的内存块的大小size，然后将其头部和尾部的分配标志位通过PACK宏设置为 0，表示该内存块变为空闲状态，最后调用coalesce函数来尝试合并该空闲内存块与相邻的空闲内存块（如果存在），进行空闲内存空间的整合。

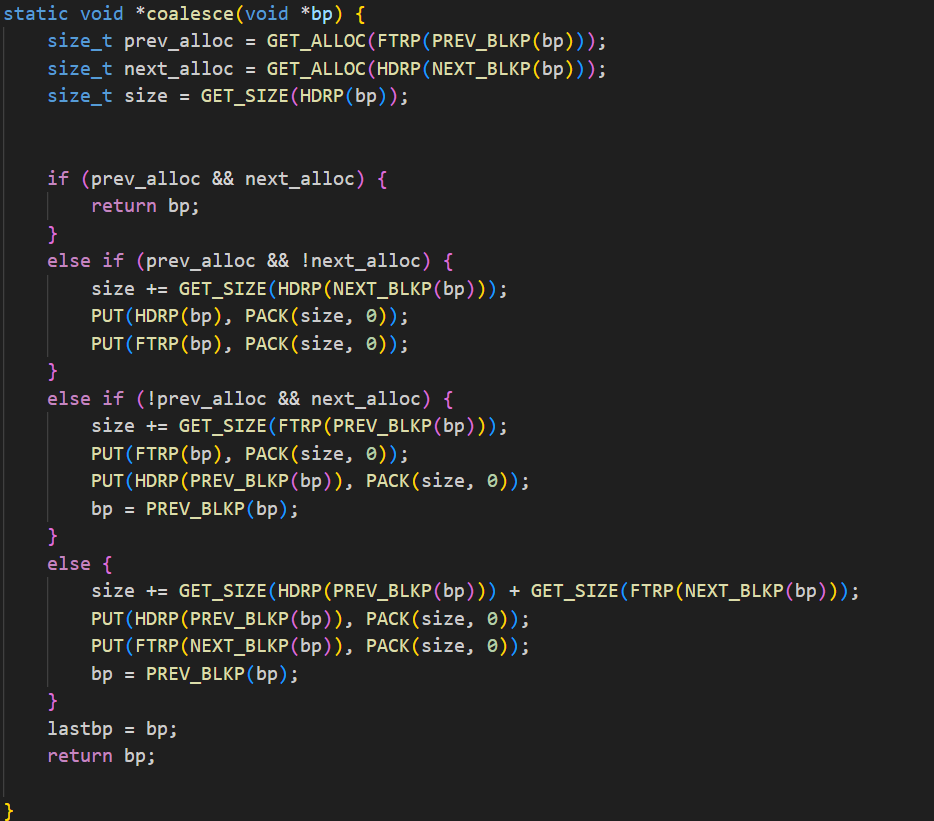


coalesce函数：首先获取当前要合并的内存块的前一个内存块的分配标志（prev\_alloc）和后一个内存块的分配标志（next\_alloc）以及当前内存块自身的大小（size）。

根据相邻内存块的分配情况分四种情况处理：

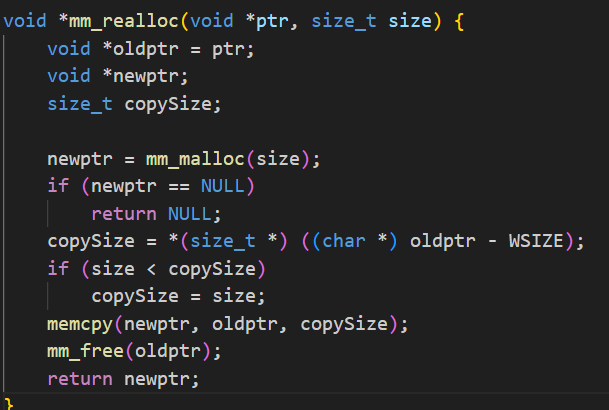
（1）、如果前后内存块都已分配（prev\_alloc和next\_alloc都为1），则无需合并，直接返回当前内存块指针bp。（2）、如果前一个内存块已分配而后一个内存块空闲（prev\_alloc为1，next\_alloc为0），则将当前内存块和后一个空闲内存块合并，更新当前内存块的大小为两者之和，并相应更新头部和尾部的大小和分配标志信息。（3）、如果前一个内存块空闲而后一个内存块已分配（prev\_alloc为0，next\_alloc为1），则将当前内存块和前一个空闲内存块合并，更新前一个内存块（通过将指针bp指向前一个内存块）的大小等信息，同样更新头部和尾部相关内容。（4）、如果前后内存块都空闲，则将当前内存块与前后两个空闲内存块合并，更新合并后的内存块（指针bp指向前一个内存块）的大小等信息，设置对应的头部和尾部标志和大小字段。

最后更新lastbp指针为当前合并后的内存块指针，并返回该指针。



mm\_realloc函数：首先记录传入的旧内存块指针oldptr，然后调用mm\_malloc函数按照新的大小size申请新的内存块空间得到newptr，如果申请失败则返回NULL。接着获取旧内存块中的实际数据大小（通过从旧内存块头部往前偏移WSIZE字节获取存储的大小信息），将其赋值给copySize，如果新申请的大小size小于旧内存块实际数据大小，则将copySize更新为size，确保只拷贝有效的数据量。

通过memcpy函数将旧内存块中的数据拷贝到新内存块中，然后调用mm\_free函数释放旧内存块，最后返回新的内存块指针newptr。



最后一张是malloclab的评分截图

