Dynamic Memory Allocation Problem

For the solution of the Dynamic memory allocation problem, I have used the certain data structures to the ease the process of memory allocation and deallocation.

The following structures will be used in the program :

- **struct mem_space :** This structure is assigned to each memory block whether used or free. It consists two variables and four pointers.

  Two variables are:
  - ➔ size : Size of the memory block
  - ➔ used : It shows whether the block is allocated or free.

  Pointers are :
  - ➔ prev : Points to the previous allocated block of memory
  - ➔ next : Points to the next allocated block of memory
  - ➔ prev1 : Points to the previous free block of memory (Not NULL if this structure is of free memory block otherwise NULL)
  - ➔ next1 : Points to the next free block of memory (Not NULL if this structure is of free memory block otherwise NULL)

Size	Used	Prev	Next	Prev1	next1

- **Memory pool (void *mp) :** This is an space which is initialized in the **MemInit()** function which will be used as the memory pool of which the memory will be allocated and deallocated.

- **Hash Table :** This hash table is used to store the free block of memory according to a hash funtion based on its size. The hash function used in this algorithm is greatest integer ($\log_2$ (size)) . Each entry of this Hash-Table points to list of free spaces. These free spaces have been added using the above mentioned hash-function.
  Eg : A free space of size 16 to 31 . It will be mapped to position x = 4 in the Hash-Table. Hence we can refine our search to the space of 2^n sizes.

2^0-2^1	2^1-2^2	2^2-2^3	2^3-2^4	…….	2^{n-1}-2^n

Algorithm :

The basic approach for solving dynamic memory allocation problem is to form a list of free blocks and search through this list for the first free block of big enough size to the requested size. This is called the first-fit algorithm. I have implemented a hash-table data structure to optimize this algorithm.

void MemInit() : Allocate a 160 KB of memory to the memory pool of which the memory block will be allocated and deallocated. It also initializes the hash-table with all entries pointing to NULL except the last entry pointing to a list containing one element of a free block of memory of size 160 KB.

MyMalloc(int bytes) : This function would allocate a block of memory to a request of memory of particular size. It looks at the particular entry in the hash table if it finds a block of memory of that size it allocates it or searches further down the list for a free block of larger size and allocates it. It also adds an entry in the hash table for the left-over free block which is (size – size_requested – sizeof(struct mem_space)). Whenever it finds a free block of memory it also assigns to it a **mem_space struct** to store the information of that block. This function returns a pointer to the starting address of free memory block.

MyFree(void *p) : This function would deallocate the allocated block of memory with a pointer p pointing to the starting address of the allocated memory. It checks in the **struct mem_space** assigned to the block and frees that block of memory. If that block of memory that is to be freed is beside a free block then these block are coalesced into one and an entry for the combined block is added to the hash-table according to the hash-function.

Order analysis :

In memory allocation, it would take O(1) (amortized cost) time.
In memory deallocation, it takes O(1) (amortized cost) with some overhead in coalescing.