```
#include <assert.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

#define HEADER_SIZE sizeof(struct chunk_header)
#define FALSE 0
#define TRUE 1
typedef int bool;


//NOTE Heap starts at size 0;
void *Heap = NULL;

struct chunk_header {
  size_t Size;
  struct chunk_header *Next;
  bool Free;
};

struct chunk_header*
FindFreeChunk(struct chunk_header **Previous, size_t Size)
{
  struct chunk_header *Current = Heap;
  while(Current && !(Current->Free && Current->Size >= Size))
    {
      *Previous = Current;
      Current = Current->Next;
    }
  return Current; // Will point to NULL if no free chunk available
}


// HELPER FUNCTION FOR ALLOCATING NEW SPACE ON THE HEAP
struct chunk_header*
RequestSpace(struct chunk_header *Previous, size_t Size)
{
  struct chunk_header *NewChunk;
  NewChunk = sbrk(0); //set Chunk to current break
```

```
  void *HeapSpaceRequest = sbrk(HEADER_SIZE + Size); //expand heap

  // Check for sbrk failure
  if(HeapSpaceRequest == (void *) -1)
    {
      return NULL;
    }

  if(Previous)
    {
      Previous->Next = NewChunk;
    }
  NewChunk->Size = Size;
  NewChunk->Next = NULL;
  NewChunk->Free = FALSE;

  return NewChunk;
}

void
Free(void *Memory)
{
  if(!Memory)
    {
      return;
    }
  struct chunk_header *Header = (struct chunk_header *) Memory - 1;
  assert(Header->Free == 0);
  Header->Free = 1;

}

void*
Alloc(size_t Size)
{

  /* CASE 1  invalid Size arguement */
  if(Size <= 0)
    {
      return NULL;
```

```
    }

  struct chunk_header *Chunk = NULL;

  /* CASE 2 The Heap has not been initialized, need to make room on the heap */
  if(!Heap)
    {
      Chunk = RequestSpace(NULL, Size);
      if(!Chunk)
{
  return NULL; // sbrk failed
}
      Heap = Chunk; // INITIALIAZE CHUNK
    }

  /* CASE 3, Heap already intialized search for free chunk  */
  else
    {
      struct chunk_header *Previous = Heap;
      Chunk = FindFreeChunk(&Previous, Size);

      if(!Chunk) //failed to find free block
{
  Chunk = RequestSpace(Previous, Size);
    if(!Chunk) // sbrk failed
      {
return NULL;
      }
}
    }
   return (Chunk + 1);
}

void main(void)
{
  int *x = (int *)Alloc(16);
  int *y = (int *)Alloc(8);
  int *z = (int *)Alloc(16);
  Free(y);
  int *k = (int *)Alloc(4);
```

```
}
```