void *a;
int *p;
p = a √
a = p ×
a = (void *)p

void *a = 1000;
int *p = 2000;
p+1 → 2004
a+1 → 2001
*(int *)a = 12; √
*p = 12 ×

int *p = malloc(100);
Assignment 5: Implementing memory allocation

```c
void* Mem_alloc(int size) {
    // 1 -> align to 4 bytes
    // 2 -> find best-fit block
    // 3 -> split the block
    // 4 -> return pointer to payload

    size = size aligned to 4-bytes

    // find the best-fit block
    block *best_block = list_head;
    for each block B in block_list {
        if (B is not allocated and
            B is bigger than size and
            B is smallest so far
            (smaller best_block))
            best_block = B;
    }
```
if no best block found:
    return error \( \rightarrow \) (Out of memory)

// split the best block
block* B = best_block

if enough space in B for another block \( E \)
    \( \rightarrow \) if \( (B\text{.size} - \text{size} > ??) \)
    \( \rightarrow \) size w/ 4 (min payload)
    \( \rightarrow \) header + sizeof(block-header)

(block* B\_new = B + sizeof(header) + size)

3 return (void*) B + sizeof(header)
int free (void * ptr)
// check for errors
if (ptr == NULL) return error
block_header * B = ptr - sizeof(header);
if (B->next->allocated) return error
set B.allocated to false
// coalesce free blocks
for each block C in block list E
  if C.free and C.next.free
    coalesce C and C.next
else
  go to next C
3
return no error
if you coalesce C
  C.next. Re-run the loop on new C

freeing