MP2 Design Document

1. FrameList(frame_list.C/H)

```
You, last week | 1 author (You)
class FrameList
{
public:
    You, last week | 1 author (You)
    struct FreeBlock
    {
        unsigned long start;
        unsigned long length;
        FreeBlock *next;
    };

    // Default constructor: no free blocks
    FrameList() : head(nullptr) {}
```

   a. Data structure for linked list
   b. Manages a list of free memory frame blocks
   c. Provides basic operations for allocation and deallocation
   d. Maintains continuous frame sequences
   e. Key operations:
      i. init_managed_region: init a free list with the whole block
      ii. allocate: find and allocate continuous frames
      iii. release: return unused memory to list and merge free block to larger block
      iv. mard_unavailable: mark frames as unavailable

2. ContFramePool(cont_frame_pool.C/H)

```
class ContFramePool
{

private:
  /* -- DEFINE YOUR CONT FRAME POOL DATA STRUCTURE(s) HE
  // Structure to track allocated frames and their sizes
  You, last week | 1 author (You)
  struct AllocatedBlock
  {
    unsigned long frame_no;
    unsigned int length;
    AllocatedBlock *next;
  };

  FrameList free_list;

  unsigned long base_frame_no;
  unsigned long nframes;
  unsigned long info_frame_no;

  // Add pointer for pool list
  ContFramePool *next_pool;
  static ContFramePool *first_pool; // Head of pool list

  AllocatedBlock *allocated_blocks;

  unsigned long allocated_block_offset;
```

a. Frame pool management
b. Supports several independent pools
c. Tracks allocated blocks
d. Manages metadata storage

3. Key functions explanation
    a. **FrameList::allocate(unsigned long n_frames)**
        i. This is a typical linked list operation.
        ii. Find the block that can fit n_frames with prev and current pointers.
        iii. If finding it, compare n_frames with current block size
            1. If fit, assign it and make prev pointing to the next of current
            2. If current is larger, assign it and reduce the available size of current.
        iv. If not finding it, always loop to the end of the linked list.
        v. Finally not finding it, return error.

b. **FrameList::release(unsigned long start, unsigned long length)**
    i. This is a merge operation in a sorted linked list.
    ii. Find the position to insert the released block using prev and current pointers.
    iii. If finding position (where start < curr->start), check three cases:
        1. Case 1: Merge with previous block
            a. If prev exists and (prev->start + prev->length == start)
            b. Add length to prev->length
            c. If can also merge with current block
            d. Add current's length to prev
            e. Make prev skip over current in the list

        2. Case 2: Merge with current block
            a. If current exists and (start + length == curr->start)
            b. Create new block with:
                i. start from released block
                ii. combined length
                iii. current's next pointer
            c. Insert this block where current was

        3. Case 3: No merging possible
            a. Create new block with released parameters
            b. Insert between prev and current in the list
            c. Special case: if no prev, make it the new head
    iv. Always maintain sorted order by start frame number.

c. **ContFramePool::ContFramePool(unsigned long _base_frame_no,unsigned long _n_frames,unsigned long _info_frame_no)**
    i. Constructor for ContFramePool
    ii. Calculate start position, saved blocks for metadata
    iii. Init a pointer to track allocated blocks

d. **ContFramePool::get_frames(unsigned int _n_frames)**
    i. This is a two-step allocation process: get frames and record allocation.
    ii. Step 1: Get frames from free_list
        1. Call free_list.allocate(n_frames) to get starting frame
        2. If allocation fails (returns 0), return failure

iii. Step 2: Record the allocation in metadata
    1. Create new AllocatedBlock in info frame area:
        a. Calculate position using info_frame_no and current offset
        b. Store frame number (relative to base_frame_no)
        c. Store length (n_frames)
        d. Link to existing allocated_blocks list
iv. Update allocated_block_offset for next allocation
v. Return absolute frame number to caller

e. ContFramePool::release_frames(unsigned long first_frame_no)

i. This is a static function that must find the owner pool before releasing.
ii. Step 1: Find owning pool
    1. Start at first_pool
    2. For each pool in linked list:
    3. Calculate pool's frame range
    4. If first_frame_no is in range
    5. This is the owner pool
iii. Step 2: Find allocation record in owner pool
    1. Search allocated_blocks list
    2. Convert first_frame_no to relative number
    3. Find matching frame_no in list
    4. Save the length needed
    5. Remove block from allocated list
iv. Step 3: Return frames to free list
    1. Call free_list.release with:
        a. absolute frame number
        b. saved length
v. If pool not found or block not found
vi. Print warning message