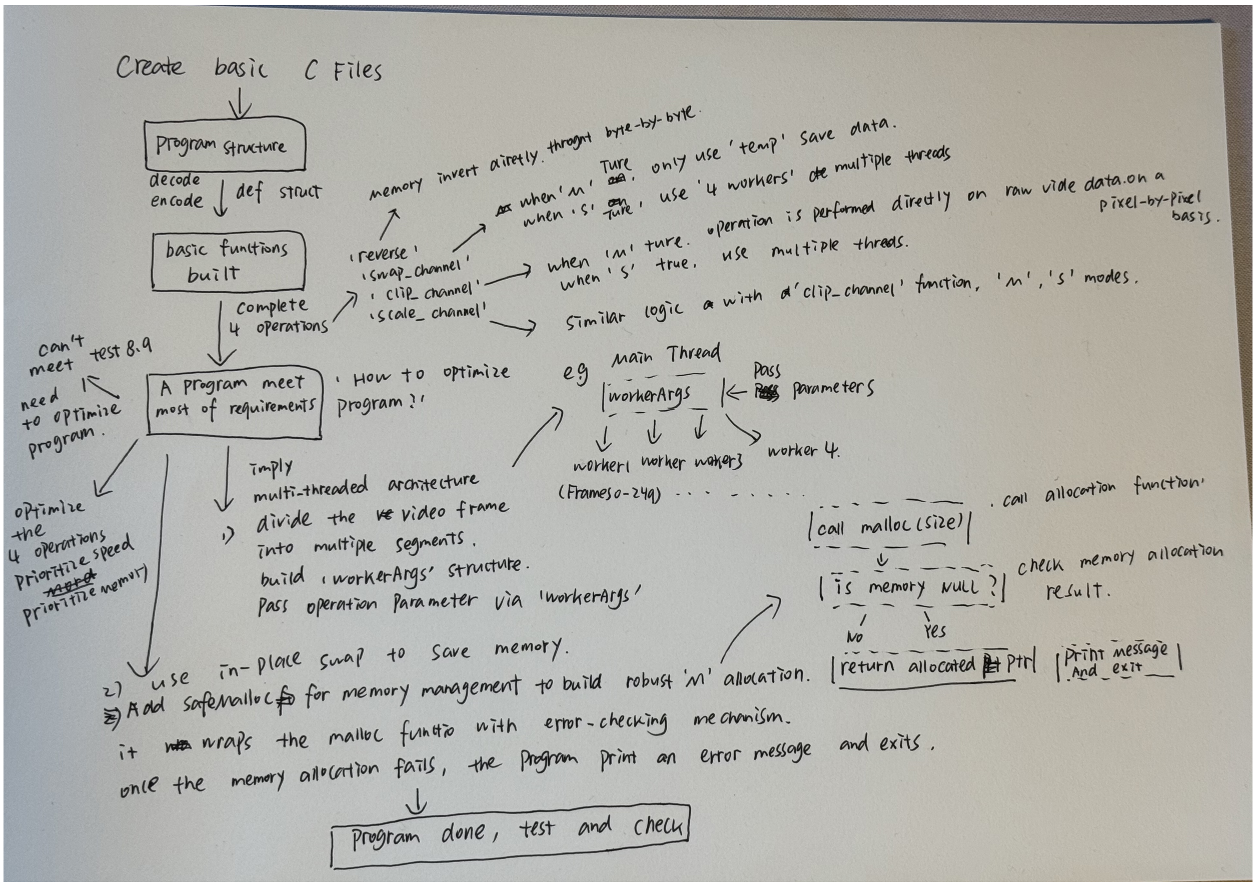
Initial Concept

The initial design of the system, createthe basic document structure, define the structures, implement the ' decodevideo', 'encodevideo' functions, the initial goal was processing the video frame operations , video inversion (REVERSE), channel swapping (SWAP), pixel clipping (CLIP) and pixel scaling (SCALE). Initially the system used a single-threaded approach, processing each frame sequentially.Then subsequently found that the performance is not enough to start adding safemalloc method to improve the stability of the program when the memory distribution, in the original function based on the modification of the use of more in-place swapping to save memory, an example of a reverse directly through the byte-by-byte invert to save the contents of and the use of multi-threaded (pthread) method of allocating frames to four threads to speed things up, and add Priorityispeed and priorityimemoryare to choose whether to prioritize memory or speed.

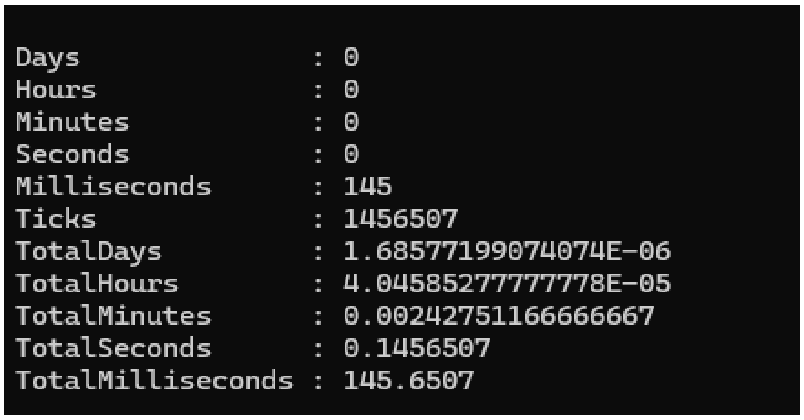

Create basic C Files
Program structure
decode encode
def struct
basic functions built
complete 4 operations
'reverse'
'swap_channel'
'clip_channel'
'scale_channel'
memory invert airetly. thrognt byte-by-byte.
when 'm' Ture, only use 'temp' save data.
when 's' Ture, use '4 workers' multiple threads
when 'm' ture. operation is performed directly on raw vide data.on a pixel-by-pixel basis.
when 's' true, use multiple threads.
Similar logic with 'clip_channel' function, 'm','s' modes.
can't meet test 8.9
need to optimize program.
A program meet most of requirements
'How to optimize program?'
optimize the 4 operations
Prioritize speed
~~Meme~~
prioritize memory
imply multi-threaded architecture
1) divide the video frame into multiple segments.
build 'workerArgs' structure.
Pass operation Parameter via 'workerArgs'
eg Main Thread
workerArgs
Pass parameters
worker1
worker
worker3
worker 4.
(Frames 0-249) . . . .
call malloc(size)
. call allocation function.
is memory NULL ?
check memory allocation result.
No
Yes
return allocated Ptr
Print message And exit
2) use in-place swap to save memory.
3) Add safeMalloc for memory management to build robust 'm' allocation.
it wraps the malloc functio with error-checking mechanism.
once the memory allocation fails, the Program print an error message and exits.
Program done, test and check

Approach to the problem and the success of the solution.
In previous versions the program didn't meet the speed and memory requirement, add a multi-threaded approach （pthread） to speed optimization by defining the WorkerArgs structure, where the WorkerArgs are created by the main thread and passed through the FrameWorker to each thread's worker function. Each thread reads the shared arguments from the FrameWorker, but only processes the frames it is responsible for, resulting in a significant speedup.
The advantages of this method are advantages:
Significantly improves the processing performance of multi-frame video operations.
Reduced data replication overhead by sharing data between threads.
Flexible task allocation and thread management allow the system to fully utilize multi-core processors.

For the memory, the safemalloc method is used to enhance the stability of memory allocation. safeMalloc is used to ensure the safety of memory allocation. malloc returns NULL in case of memory allocation failure. In the case of memory allocation failure, malloc will return NULL, if the program directly use this NULL pointer, it will lead to undefined behavior (such as program crash). safeMalloc by adding error checking and error handling mechanism, to solve this problem, for the problem of memory occupation, I use more in-place swapping, to reduce the memory of the For the memory footprint issue, I used more in-place swapping to reduce the memory footprint, and used logic that operates directly on the data in the operation's function to reduce the memory footprint.
The advantages of the method:
Can prevent crashes If a memory allocation fails, safeMalloc detects it in advance and exits the program in a safe way, rather than causing undefined behavior.
Improves coderobustness by ensuring that programs do not crash at runtime due to memory allocation problems, especially in environments dealing with large data or memory constraints.
The logic of safeMalloc can be further extended to facilitate debugging by, for example, logging contextual information (e.g., time, function name, etc.) in the event of an allocation failure.
Reduce memory footprint by logic that operates directly on read data.

Show performance：

After testing the code memory occupation is significantly reduced, the running speed is accelerated, a windows system to test the running speed of the data as shown in the figure, the test file for the test.bin file.



```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 145
Ticks             : 1456507
TotalDays         : 1.68577199074074E-06
TotalHours        : 4.04585277777778E-05
TotalMinutes      : 0.00242751166666667
TotalSeconds      : 0.1456507
TotalMilliseconds : 145.6507
```

summary：

Running speed:
The use of multi-threaded workload distribution through pthread enables efficient parallel processing, especially in channel swapping, cropping and scaling operations.
Different optimization paths prioritizing Speed and Memory are supported for diverse scenarios.

Memory optimization:
Reduces the risk of high memory usage by sharing FrameWorker and batch operation of video frames in multi-threaded mode.
Provide memory-safe allocation functions to prevent program crashes due to memory allocation failures.
Different implementation strategies are designed for different optimization requirements (speed,memory) to flexibly adapt to different operating environments.

Advanced Features and Architecture
Using pthread to realize multi-thread parallel processing (e.g. startWorkers and worker functions), multiple frames can be processed at the same time, which significantly improves the processing speed.
Efficient load distribution is achieved by splitting tasks across threads (e.g. FrameWorker assigns frame ranges and operation types).
Organize task parameters using nested structures and unions, such as the WorkerArgs structure: Flexible priority modes:
Provides three optimization modes (prioritizeMemory, prioritizeSpeed and default mode):
prioritizeSpeed: uses memory block operations (e.g. memcpy) and multithreading.
prioritizeMemory: uses pixel-by-pixel operations to minimize additional memory usage.
Default mode: compromise processing.
Allows the user to choose different performance optimization strategies based on requirements.
Wraps memory allocations with safeMalloc to ensure safety and error handling for memory allocation failures.