C++ Standard I/O Library – stdio.cpp Documentation

Author: Aldan Brown
Course: CSS 503
Date: May 22, 2025

Execution Output – Diff Tests

```
[aldanb@csslab8 Program3]$ script diffTest
Script started, output log file is 'diffTest'.
[aldanb@csslab8 Program3]$ ./compile.sh
[aldanb@csslab8 Program3]$ ./driver hamlet.txt > output_hamlet.txt
[aldanb@csslab8 Program3]$ ./driver othello.txt > output_othello.txt
[aldanb@csslab8 Program3]$ diff output_hamlet.txt Originals/output_hamlet.txt
[aldanb@csslab8 Program3]$ diff output_othello.txt Originals/output_othello.txt
[aldanb@csslab8 Program3]$ diff test1.txt Originals/test1.txt
[aldanb@csslab8 Program3]$ diff test2.txt Originals/test2.txt
[aldanb@csslab8 Program3]$ diff test3.txt Originals/test3.txt
[aldanb@csslab8 Program3]$ exit
exit
Script done.
```

Execution Output – Eval Tests

```
[aldanb@csslab8 Program3]$ script evalTest
Script started, output log file is 'evalTest'.
[aldanb@csslab8 Program3]$ ./compile.sh
[aldanb@csslab8 Program3]$ ./eval_tests.sh > eval_test_out.txt
[aldanb@csslab8 Program3]$ ./compile.sh
[aldanb@csslab8 Program3]$ ./eval_tests.sh > eval_test_org_out.txt
[aldanb@csslab8 Program3]$ exit
exit
Script done.
```

```
1   Reads : Unix   I/O [Read   once     ] = 187
2   Reads : Unix   I/O [Block  transfers] = 126
3   Reads : Unix   I/O [Char   transfers] = 218710
4   Reads : Unix   I/O [Random transfers] = 260
5   Reads : C File I/O [Read   once     ] = 256
6   Reads : C File I/O [Block  transfers] = 82
7   Reads : C File I/O [Char   transfers] = 1550
8   Reads : C File I/O [Random transfers] = 118
9   Writes: Unix   I/O [Read   once     ] = 177
10  Writes: Unix   I/O [Block  transfers] = 127
11  Writes: Unix   I/O [Char   transfers] = 27405376
12  Writes: Unix   I/O [Random transfers] = 68049
13  Writes: C File I/O [Read   once     ] = 170
14  Writes: C File I/O [Block  transfers] = 169
15  Writes: C File I/O [Char   transfers] = 1617
16  Writes: C File I/O [Random transfers] = 209
17
```

“stdio.h”

```
1   Reads : Unix   I/O [Read   once     ] = 144
2   Reads : Unix   I/O [Block  transfers] = 229
3   Reads : Unix   I/O [Char   transfers] = 218418
4   Reads : Unix   I/O [Random transfers] = 263
5   Reads : C File I/O [Read   once     ] = 205
6   Reads : C File I/O [Block  transfers] = 91
7   Reads : C File I/O [Char   transfers] = 879
8   Reads : C File I/O [Random transfers] = 92
9   Writes: Unix   I/O [Read   once     ] = 92
10  Writes: Unix   I/O [Block  transfers] = 173
11  Writes: Unix   I/O [Char   transfers] = 25479982
12  Writes: Unix   I/O [Random transfers] = 163960
13  Writes: C File I/O [Read   once     ] = 109
14  Writes: C File I/O [Block  transfers] = 154
15  Writes: C File I/O [Char   transfers] = 876
16  Writes: C File I/O [Random transfers] = 180
17
```

<stdio.h>

Implementation

1. File Structure Management
    - A custom FILE structure is defined, buffer, buffer position, mode (_IONBF, _IOLBF, _IOFBF), and last operation (read/write)
    - File open modes (r, w, a, and variations) are translated into Unix flags
2. Formatted Output
    - printf() implementation supports printing integers
    - Uses itoa() to convert integers to character arrays and writes them directly to stdout via write()
3. Buffering Support
    - setvbuf() and setbuf() allow control over buffering modes:
        - _IONBF: no buffering
        - _IOFBF: full buffering
    - Buffered writes and reads use internal memory and delay system calls until buffer is full or flushed
4. File Operations
    - fopen(), fclose(): wrap Unix open() and close() to create and destroy FILE structures
    - fread(), fwrite(): read and write data with support for buffering
    - fseek(): move file position using lseek()
    - fflush(), fpurge(): manage internal buffers and flush data to disk or clear internal states
    - feof(): detect end-of-file status
5. Character and String I/O
    - fgetc(), fputc(): read/write one character at a time
    - fgets(), fputs(): read/write entire strings (lines) with basic newline handling

Limitations and Extensions

As compared to the standard library features, my implementation is much more stripped down. The following are not included in my version and could be extended (note that this list is not exhaustive):

- Only %d is supported is supported in printf(). No support for %s, %f, %x, etc
- No support for ferror() or perror() to inspect stream errors
- Not thread safe and does not have flockfile()/funlockfile() or internal mutex protection
- No handling of wchar_t, fgetwc, fputwc, etc
- No ungetc(), can't push characters back onto the input stream

There are some code design and best practice issues that should be fixed as well:

- Some memory allocations (in itoa) use new[] without corresponding delete[] or free(), leading to memory leaks in client code if not handled properly
- The stdio.h header does not follow guidelines for proper design and should be fixed to adhere to proper standards:
    - Uses "includes" in the header rather than the implementation file (backwards and does not translate to other programs well). Can cause issues when "using namespace std;" between different programs including ambiguous errors and multiple implementations when compiling.
    - No function declarations in the header. The consequence is that if recursive_iota in stdio.cpp is placed below iota, the program won't compile
        - Reduces the ability to organize in a better system for readability
    - Includes implementation code in the header (constructor)
    - No encapsulation, all data is public
    - This also means the .cpp code would need to be refactored to match. Eval.cpp and driver.cpp would need some code changed as well to use this new structure

Performance Considerations

“stdio.h” vs. Unix I/O

- My program is buffered by default, improving performance over Unix I/O for large task
- Slower than Unix I/O for small, frequent reads due to function call overhead
- Better performance in batch processing due to buffering
- Unbuffered mode (_IONBF) matches raw Unix I/O in speed (direct reading/writing)

“stdio.h” vs. <stdio.h>

- As expected, my implementation was slower across the board BUT only by a factor of tens-of-microseconds in some of the shorter read/write activities
- Speed difference would only really be noticeable on the larger file structures
- Any improvements can probably be explained by my lack of thread locking/unlocking and any additional error checking pre-built in the standard library