Assignment 7 – XXD

Vybavnag Kandasamy

CSE 13S – Fall 2023

Purpose

The program is named xd, is designed to display binary files in a human-readable hexadecimal format. This functionality is essential in scenarios where understanding the content of binary files is necessary, and a straightforward, readable format is required. In its simplest form, xd converts the binary data into hexadecimal representation, making it easier for users to interpret the content of these files.

How to Use the Program

To use the xd program, the user should first compile it using the provided makefile. Once compiled, xd can be run from the command line. If no arguments are provided, xd reads from the standard input (stdin) and outputs to the standard output (stdout). If a filename is supplied as an argument, xd reads from that file.

to run use the code provided below after ensuring all required files, xd.c, bad_xd.c and makefile are present in working directory.

```
make

./xd inputfile.txt or ./bad_xd inputfile.txt

make clean
```

Program Design

The xd program is structured around key components for processing binary files. The main data structure used is a buffer that reads and stores 16 bytes at a time. This buffer is crucial for converting binary data into hexadecimal format efficiently. The primary algorithm involves reading binary data, converting it to hexadecimal, and formatting the output appropriately. The code is organized to facilitate maintenance and future modifications, ensuring that someone familiar with the program can easily understand and modify its components.

I anticipate encountering errors if the file is invalid or if there are formatting issues. To address these, I intend to either display error messages or simply terminate the code if the file is invalid or the formatting doesn't match expectations.

Algorithms

pusdeocode for xd.c:

```
Include Standard IO Library

Function printHexAndAscii(buffer, bytesRead, offset):
    Print offset in hexadecimal
    Loop over each byte in buffer:
        Print byte in hexadecimal
    Print ASCII representation of bytes
End Function

Main Function:
    If incorrect number of arguments:
        Print usage instructions and exit

    Open file in binary mode
    If file opening fails, show error and exit

    While there's data to read:
        Read data into buffer
        Call printHexAndAscii with buffer, bytes read, and current offset
        Update offset

    Close file
End Main Function
```

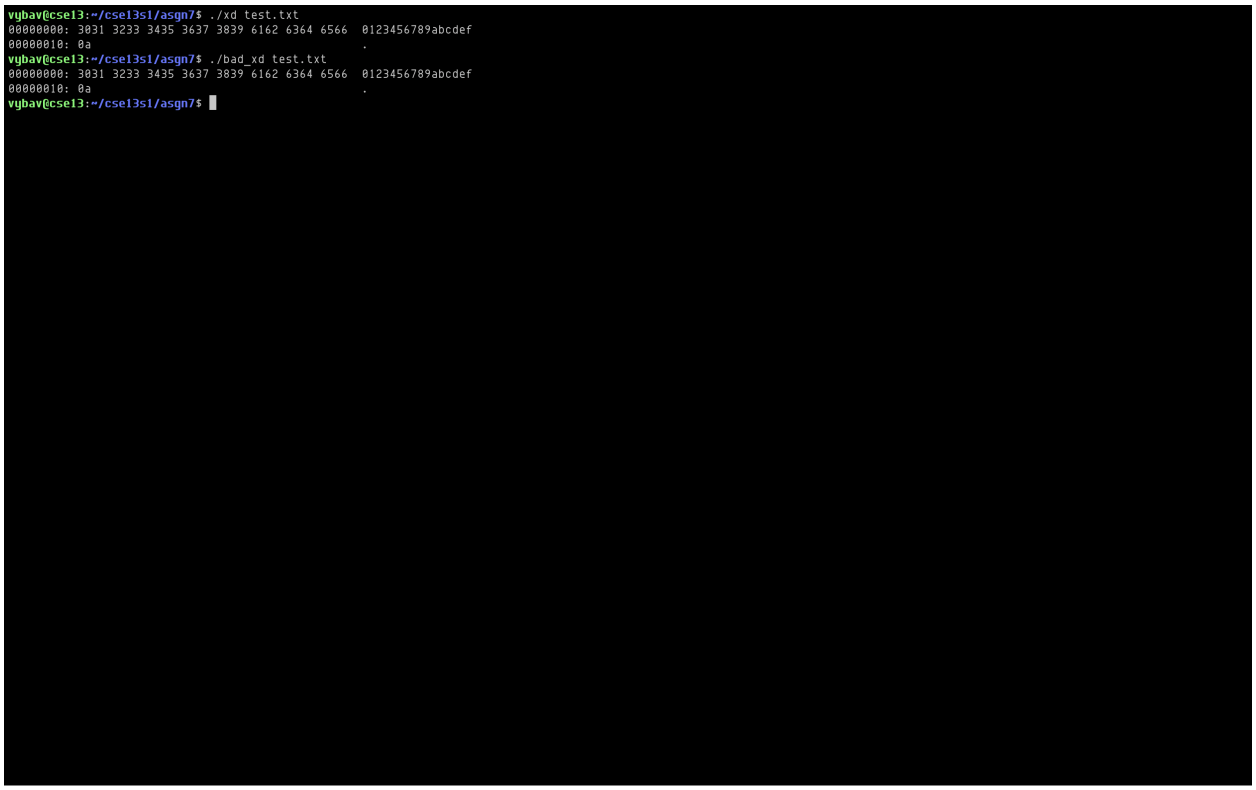
Results

To make the code shorter, I did a few simple things. First, I used shorter names for things. For example, I changed printHexAndAscii to just ph. This made the code look cleaner and take up less space. Then, I used something called a ternary operator, which is a shorter way to write if-else statements, making the main function simpler. I also used a trick with define to replace putchar with just P. This meant I could use P instead of putchar, saving a lot of space.

I added some new parts to the code, like errno.h and unistd.h. These are like toolboxes that help with things like reading files and handling errors, which made the code more efficient. Instead of using printf a lot, I used putchar or P for showing characters, which is shorter and simpler.

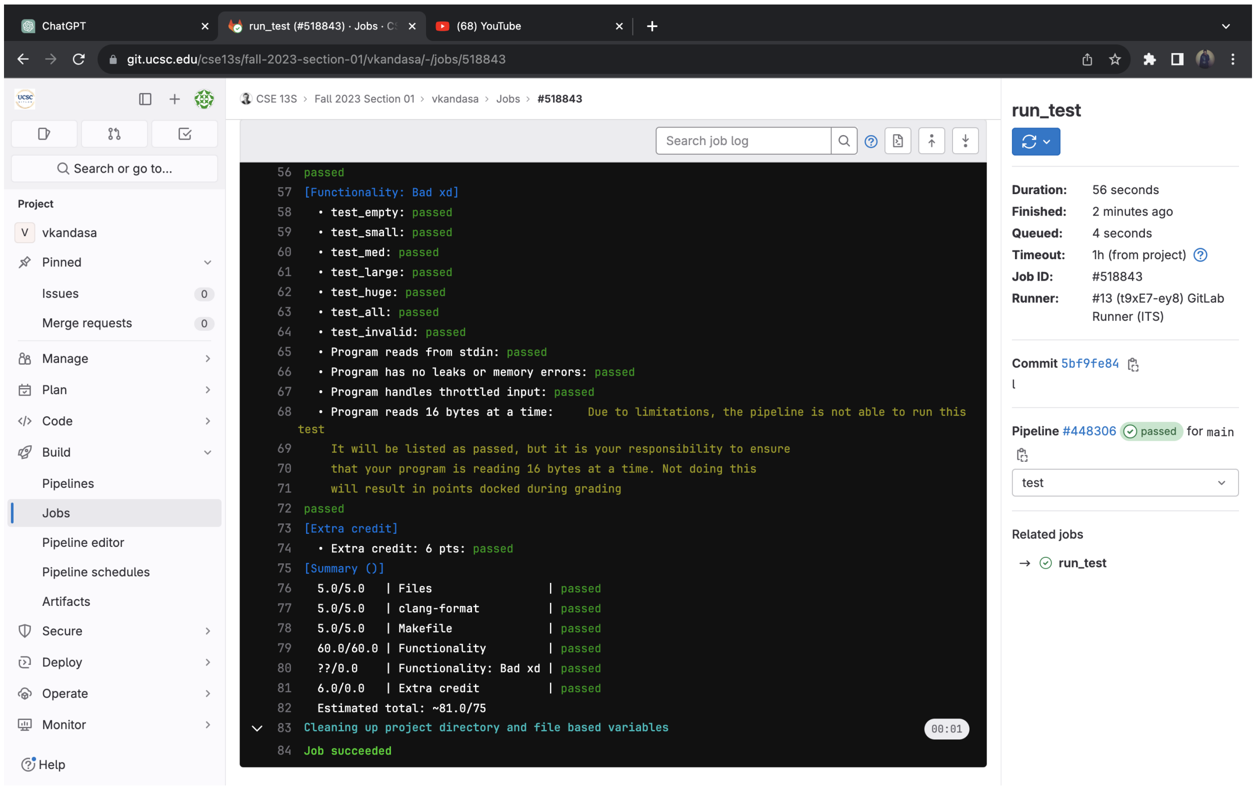
All these changes made the code much smaller. The original code, called xd.c, was easier to read and understand. The new version, bad_xd.c, was focused on being short. By using short names, the ternary operator, and define for putchar, along with the new toolboxes, I cut down a lot of the code. In the end, the new code was only 906 characters long. This is a big difference from the original and shows how these simple tricks can make code much shorter.

Here are the screenshots the files running:



```
vybav@cse13:~/cse13s1/asgn7$ ./xd test.txt
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 0a                                       .
vybav@cse13:~/cse13s1/asgn7$ ./bad_xd test.txt
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 0a                                       .
vybav@cse13:~/cse13s1/asgn7$ 
```

Figure 1: Screenshot of the program running.



```
passed
[Functionality: Bad xd]
  • test_empty: passed
  • test_small: passed
  • test_med: passed
  • test_large: passed
  • test_huge: passed
  • test_all: passed
  • test_invalid: passed
  • Program reads from stdin: passed
  • Program has no leaks or memory errors: passed
  • Program handles throttled input: passed
  • Program reads 16 bytes at a time:     Due to limitations, the pipeline is not able to run this
test
    It will be listed as passed, but it is your responsibility to ensure
    that your program is reading 16 bytes at a time. Not doing this
    will result in points docked during grading
passed
[Extra credit]
  • Extra credit: 6 pts: passed
[Summary ()]
  5.0/5.0   | Files                 | passed
  5.0/5.0   | clang-format          | passed
  5.0/5.0   | Makefile              | passed
  60.0/60.0 | Functionality         | passed
  ??/0.0    | Functionality: Bad xd | passed
  6.0/0.0   | Extra credit          | passed
  Estimated total: ~81.0/75
Cleaning up project directory and file based variables                00:01
Job succeeded
```

run_test

Duration: 56 seconds
Finished: 2 minutes ago
Queued: 4 seconds
Timeout: 1h (from project)
Job ID: #518843
Runner: #13 (t9xE7-ey8) GitLab Runner (ITS)

Commit 5bf9fe84
l

Pipeline #448306 passed for main

test

Related jobs
→ run_test

Figure 2: Screenshot of the pipeline.