Assignment 7 – XXD

Sebastian Avila

CSE 13S – Fall 2023

Purpose

The purpose of this program is to make a simple version of the `xxd` command. This program takes input and prints out the index value of the first byte, the hexadecimal values of each character, and the input read. It does this with a buffer size of 16 bytes.

How to Use the Program

To use this program you will need to have two files in the same directory: `xd.c` and `Makefile`. With these files in the same directory, use command `make clean` to remove any preexisting object files. Then, use command `make format` to format all the header and source files. Then use command `make` to compile the program. You may also have an input file in the directory or in a sub-folder. Then run the program using `./xd -i inputfile` or `./xd`. The latter uses `stdin` as input. The program only either accepts one command line argument or none. If more than one is given, the program terminates. The program uses several optional compiler flags:

- `-Wall`: This flag enables all warning messages.
- `-Werror`: This flag turns all warnings into errors.
- `-Wextra`: This flag enables extra warning flags that are not enabled by `-Wall`.
- `-Wconversion`: This flag warns for any implicit conversion that may change a value.
- `-Wdouble-promotion`: This flag warns when a float is implicitly promoted to a double.
- `-Wstrict-prototypes`: This flag warns if a function is declared or defined without specifying the argument types.
- `-pedantic`: This flag issues all the warnings demanded by strict ISO C and ISO C++.

Program Design

The program begins by checking the command line arguments. If their are more than 2 (the program executable and an input file), then the program exits with a nonzero value. Otherwise, it creates a integer variable to hold the input file descriptor value. It sets this value to `stdin`. It then checks if the number of arguments is equal to 2; if it is, it attempts to open a file with the name of the second argument. If the file does not open successfully, the program terminates. If the number of command line arguments is not 2, then `stdin` remains as the input file. The program then initializes an empty character string of length 17 for the buffer. It also initializes an integer value to hold the current index of the first byte. It then reads from the input file and store the number of bytes read in a variable `res`. It then begins a loop which continues until exited. In this loop, it first checks if 0 bytes were read. If this is the case, the program terminates. Otherwise, it checks if 16 bytes were read. If 16 bytes were read, the program first prints the index as a hex value, then it prints the hex values of the characters in the buffer using a function, then it prints the

buffer. It then sets all the bytes in the buffer string to the null character, sets the number of bytes read to 0 and increases the index by 16. It then checks whether `res` is equal to -1. This means an error occurs while reading input. If this happens, the program terminates. Otherwise, it reads input again and stores the number of bytes into a variable `temp`. If 0 bytes are read, the program breaks out of the loop. If an error occurs while reading, the program terminates. Otherwise, it adds the number of bytes read in `temp` to `res`. It then restarts the loop. Once the loop ends, the program checks if the number of bytes read is greater than 0. If it is, it prints the index as a hex value, then it prints the hex values of the characters in the buffer using a function, then it prints the buffer. It then returns 0 and terminates.

Function Descriptions

void print_hex(char *buffer, int len)

This function takes in two parameters: a pointer to a character string `buffer` and an integer `len`. It does not return anything. Its purpose is to print each of the characters as their corresponding hex value with two digits. The values are printed in groups of 4 digits with a space in between. If the buffer does not contain 16 bytes, spaces are printed instead of the hex value for any empty bytes.

```
void print_hex(char *buffer, int len)
    for i, 1 to i <= 16
        if i > len then
            print two spaces
        else
            print hex value with 2 digits

        if i % 2 = 0 then
            print one space

    print one space
```

Psuedocode

```
buffered reading
    char buffer[17] = ""
    res = read(fdin, &buffer, 16)

    while true
        if res = 0 then
            return 0

        if res == 16 then
            print program output
            set all bytes in buffer to "\0"

        if res = -1 then
            return 1

        temp = read(fdin, buffer + res, 16-res);
        if temp = 0 then
            break
        if temp == -1 then
            return 0

        res = res + temp

    if res > 0
        print program output
```

```
main
    if argc > 2 then
        return 1

    fdin = 0;

    if argc == 2 then
        fdin = open(argv[1], O_RDONLY, 0)
        if fdin = -1 then
            return 1

    char buffer[17] = ""
    res = read(fdin, &buffer, 16)
    index = 0

    while true
        if res = 0 then
            return 0

        if res == 16 then
            print index as hex
            print_hex(buffer,16)
            for i, 0 to i < 16
                if 32 > buffer[i] > 126 then
                    buffer[i] = "."
            print buffer
            set all bytes in buffer to "\0"
            index = index + 16

        if res = -1 then
            return 1

        temp = read(fdin, buffer + res, 16-res);
        if temp = 0 then
            break
        if temp == -1 then
            return 0

        res = res + temp

    if res > 0
        print index as hex
        print_hex(buffer,16)
        for i, 0 to i < 16
            if 32 > buffer[i] > 126 then
                buffer[i] = "."
        print buffer

    return 0
```

Error Handling

- Too many command line arguments : If too many command line arguments are given, the program will be terminated with a nonzero exit code.
- Invalid input file: If the program cannot open the input file, the program will be terminated with a nonzero exit code.

- Errors will reading program : If the program encounters any errors while reading input, the program will be terminated with a nonzero exit code.

Results

I made the program 889 characters. To shorten the program, I made all the variable names 1 character and deleted any extra lines. I also rewrote some aspects of my program. I think I found a better way to read input than how I had done in my original program. However, my original program works as intended so I did not change it. I also used the ternary operator to determine what to print rather than using multiple if-else statements. I also used commas to separate operations rather than using a semicolon and a new line. This allowed me to put multiple commands in one line which made the program shorter because there were less space characters from indentations. An example of this and the use of the ternary operator is as follows:

```
printf(i > l ? " " : "%02x", (unsigned char) b[i - 1]), printf(i % 2 == 0 ? " " : "");
```

This is two print statements in one line. Within each of the print statements, there is a ternary operation that determines what to print. The comma separator also made it possible to have if statements with no brackets that still contained multiple operations. Those actions made the most difference in the length of the program. I also used `#define` to make a macro for the `return` function. I made it a single character followed by a single character for input: `r(o)` This saved a small amount of characters but not too many. I also created a function to print the output of the program. Making a function adds a lot of characters but I had two places where the output was printed in my program, so making a function was less characters than having the code to print the output appear twice. I think refactoring my code made the most impact on shortening the program. My original implementation was a lot longer; it was 1750 characters. The psuedocode for `bad_xd.c` is below. It contains both the main function and the function to print the output: `p`.

```
bad_xd
    p(char *b, long l)
        for i, 0 to i < 16
            print(two spaces if i > l else b[i-1] as hex with 2 digits), print(one space if i % 2 =
                0 else nothing)
        print(one space)
        for j, 0 to j < l
            print(a period if 32 > b[j] > 126 else b[j])
        print(newline)

    main
        if c > 2 then
            r(1)
        f = open(v[1], O_RDONLY, 0) if c = 2 else 0, x = 0
        if f = -1 then
            r(1)
        b[17] = ""
        r = 0, t
        while true
            if (t = read(f, b + r, 16 - r)) then
                r += t
            else
                if r then
                    print(x as hex with 8 digits), p(b, r)
                r(0)
            if r < 0 then
                r(1)
            if r = 16 then
                    print(x as hex with 8 digits), p(b, r), r = 0, x += 16
```