C Programming: debugging, dynamic data structures: linked lists, patching binary files.

Objectives

- Allocating and freeing memory.
- Implementing linked lists in C.
- Basic manipulation of "binary" files.

In this lab you are required to use valgrind to make sure your program is "memory-leak" free.
You should use valgrind in the following manner: valgrind --leak-check=full --show-reachable=yes [your-program] [your-program-options]

Task 0: Memory Leaks, Segmentation Faults, and Printing data from files in hexadecimal format

Task0a:

Programs inevitably contain bugs, at least when they are still being developed. Interactive debugging using `valgrind(1)` helps locate and eliminate bugs. `valgrind` assists in discovering illegal memory access even when no segmentation fault occurs (e.g. when reading the n+1 place of an array of size n). `valgrind` is extremely useful for discovering and fixing memory leaks. It will tell you which memory allocation was not freed.

To run valgrind write: `valgrind --leak-check=full --show-reachable=yes [program-name] [program parameters]`.
If valgrind reports errors in your code, repeat the execution with the "-v" flag like so: `valgrind -v --leak-check=full --show-reachable=yes [program-name] [program parameters]`.

The source code of a buggy program, bubblesort.c, is provided. The program should sort numbers specified in the command line and print the sorted numbers, like this:

```
$ bubblesort 3 4 2 1
Original array: 3 4 2 1
Sorted array: 1 2 3 4
```

However, an illegal memory access causes a segmentation fault (segfault). In addition, the program has a few memory leaks.
First solve the segfault using `gdb` (or just by reading the code). Then use valgrind to find the memory leaks and fix them.

Task0b

Write a program that receives the name of a binary file as a command-line argument, and prints the hexadecimal value of each byte in the file in sequence to the standard output (using printf). Consult the ***printf(3)*** man page for hexadecimal format printing.

NAME

hexaPrint - prints the hexdecimal value of the input bytes from a given file

SYNOPSIS

hexaPrint FILE

DESCRIPTION

hexaPrint receives, as a command-line argument, the name of a "binary" file, and prints the hexadecimal value of each byte to the standard output, separated by spaces.

For example, your program will print the following output for this exampleFile (download using right click, save as):

```
#>hexaPrint exampleFile
63 68 65 63 6B AA DD 4D 79 0C 48 65 78
```

You should implement this program using:

- *fread(3)* to read data from the file into memory.
- A helper function, *PrintHex(buffer, length)*, that prints length bytes from memory location buffer, in hexadecimal format.

Task 1

Write a program called *bcmp* that computes differences between two binary files.

NAME

bcmp - compute differences between two binary files

SYNOPSIS

bcmp [OPTIONS] ORIG NEW

DESCRIPTION

bcmp receives two binary files ORIG and NEW and prints the list of differences between them, in the following format:

byte POSITION ORIG_VALUE NEW_VALUE

where POSITION is the position of the differing byte in the file, **starting from 0**, ORIG_VALUE is the value of the byte in ORIG, NEW_VALUE is the value of the byte in NEW. The bytes must printed in hexadecimal, as in task0.

OPTIONS

-o <output file>: Save result to an output file
-t : Print total number of differences
-k <n> : Print only first n differences

SEE ALSO

cmp
EXAMPLES

```
$> bcmp foo bar
byte 33 AE  E6
byte 35 0E  78
byte 50 FA  0C
$> bcmp -t foo bar -o out_file
$> more out_file
3
$> bcmp -k 2 foo bar -o out_file
$> more out_file
byte 33 AE  E6
byte 35 0E  78
```

Task 1a

To save the bytes that are different into memory we are going to implement a linked list of structs, and some helper functions. The diff struct is defined as follows:

```
typedef struct diff {
    long offset; /* offset of the difference in file starting from zero*/
    unsigned char orig_value;     /* value of the byte in ORIG */
    unsigned char new_value;     /* value of the byte in NEW */
} diff;
```

Each node in the linked list is represented by the following structure:

```
typedef struct node node;

struct node {
    diff *diff_data; /* pointer to a struct containing the offset and the
value of the bytes in each of the files*/
    node *next;
};
```

Implement the following functions:

```
void list_print(node *diff_list,FILE* output);
     /* Print the nodes in diff_list in the following format: byte POSITION
ORIG_VALUE NEW_VALUE.
Each item followed by a newline character. */

node* list_append(node* diff_list, diff* data);
     /* Add a new node with the given data to the list,
        and return a pointer to the list (i.e., the first node in the list).
        If the list is null - create a new entry and return a pointer to the
entry.*/
```

```
void list_free(node *diff_list); /* Free the memory allocated by and for the
list. */
```

Note: you may add the node at the beginning or end of the list, please explain your choice in terms of efficiency.

To test your list implementation you are requested to write a program that:

- Creates a list and adds 2 nodes to it.
- Prints the content of the list in the requested format.
- Frees the memory allocated for the list.

Task 1b

Now that you tested your linked list implementation, and made sure it works as requested, lets start implementing the bcmp program. The program must print the difference between the 2 files, until the end of the shortest file is reached.

- Open both files for reading.
- Find file size.
- Go over both files each time reading a single byte until the end of the shortest file is reach.
- Use the linked list implementation from Task 1a to save the differences.
- Print diff list according to the given flags.

To test your code do the following:

- Copy your executable to a file called 1: `cp bcmp 1`.
- Create another copy called 2.
- Open 2 with a hexedit: `hexedit 2`. hexedit is a program for viewing binary files in hex. Each sequence of 2 digits constitute a single byte. **LOOK LIKE HEXEDIT IS NOT INSTALLED ON THE LAB COMPUTERS? USE https://hexed.it/ INSTEAD.**
- Change a couple of bytes in the file and save the changes.
- Run your program and see if it's able to detect the changes you made.

Task 1c

Add support to the -o, -t, and -k flags.

Task 2

In this task you will implement a code that restores the NEW file back to it's ORIG, using the diff_list you created in task 1b, and fseek. The bcmp program restores the NEW file back to its ORIG if a -r flag is given. After running the program with this flag, both ORIG and NEW will be the same as ORIG, and running the program on the files, would result in no differences.

NAME

  bcmp - compute differences between two binary files

SYNOPSIS

  bcmp [OPTIONS] ORIG NEW

DESCRIPTION

  bcmp receives two binary files ORIG and NEW and prints the list of differences between them, in the following format:

  byte POSITION ORIG_VALUE NEW_VALUE

  where POSITION is the position of the differing byte in the file, **starting from 0**, ORIG_VALUE is the value of the byte in ORIG, NEW_VALUE is the value of the byte in NEW. The bytes must printed in hexadecimal, as in task0.

OPTIONS

  -o <output file>: Save differences list to an output file
  -t : Print total number of differences
  -k <n> : Print only first n differences.
  -r <n>: Restore the NEW file back to it's ORIG content. If a number n is given, restore only n first differences.

SEE ALSO

  cmp