Department of Electrical and Electronic Engineering

EEEE1042

Introduction to Software Engineering and Programming

Coursework 1 - Report

Koo Yan Jie

20620542

November 2023

Abstract

Histograms are a widely used method to represent frequency distribution. Creating a program that can plot a histogram is a very good exercise to practice managing data in C using arrays and functions. This report aims to provide an overview of the structure of the program, as well as the design decisions made when coming up with a solution. Additionally, a few problems encountered throughout, as well as the limitations of the program are discussed.

Table of Contents

1 Introduction

Histograms are a very commonly used graphical representation of frequency distributions. The main task of this coursework is to create a program using C that can extract data from a file and plot a histogram using that data. The histograms are to be printed as ASCII histograms, using entirely text-based graphics.

There are 2 main problems to solve in this coursework: reading data from a file and plotting an ASCII histogram. This report aims to address these problems, as well as explaining the design decisions behind the solutions. Additionally, the results of the program output will be discussed along with some problems encountered during the coursework, and a few limitations of the program.

2 The main( ) Function

The main function is structured in 2 different sections; the first section declares and initialises variables and arrays, and the second section is the calling environment for all subfunctions. As the program is required to take input from the command line, the main function is declared with 2 arguments. `int argc` returns the number of arguments passed and `char **argv` stores the argument string.

2.1 Variables and Arrays

```
int L = 100;

float bin[L];
float freq[L];
char *word[5*L];

int N = 0;
int W = 0;

int S = 50;
char B = 'o';
```

Fig. 1. Variables and arrays.

A variable int L is first declared to initialise the array sizes. Three arrays are then declared to store data. `float bin[L]` and `float freq[L]` allocates L number of floats to store the

histogram bin data and the corresponding frequency. `char *word[5*L]` allocates space to store 5*L number of unique strings. The size of the array for each string is not declared, as that will be allocated in a sub-function based on the length of the string as to save memory.

Another four variables are initialised to store integers used for counting and indexing, and for setting the length and style of the histogram bars. `int N` is for counting the number of bins in the histogram and is initialised to 0. `int W` counts the number of words in the .txt file and is also initialised to 0. `int S` sets the maximum length of the histogram bars, and `char B` sets the character used to print the ASCII histogram.

2.2 Calling Environment

```
if (argc>1){
      readHistFile(argv[1], bin, freq, &N , L);

      printHist(bin, freq, N);

      printf("\n===horizontal histogram===\n");
      plotHistH(bin, freq, N, S, B);

      printf("\n===vertical histogram===\n");
      plotHistV(bin, freq, N, S, B);
...
```

Fig. 2. Printing histograms from the .hist file.

The program first checks to see if there are any arguments passed by the user. If no arguments are passed, the program shows an error and terminates. If there is at least 1 argument (`argc > 1`), then the first argument (`argv[1]`) is passed to a `readHistFile` function along with pointers to the `bin` array, `freq` array and variables `N`, and `L`. The function reads the file, and if it succeeds (provided that the file exists, and is formatted correctly), it stores the contents of the file into the `bin` and `freq` array. The printHist function then prints out the contents of the `bin` and `freq` array. A horizontal histogram is plotted using `plotHistH` by passing the `bin` and `freq` array, and variables `N`, `S`, and `B` as arguments. Similarly, a vertical histogram is plotted using `plotHistV`, accepting the same arguments. After printing the histograms, a `clearHist` function is called to reset the `bin` and `freq` array, as well as the variable `N`.

```
if(argc>2){
      readTextFile(argv[2], word, &W , 5*L);

      printf("\nNumber of words in the file: %d\n", W);

      sortWordLen(word, bin, freq, W, &N);

      printf("\nPlotting histogram...\n");

      printf("\n===horizontal histogram===\n");
      plotHistH(bin, freq, N, S, B);

      printf("\n===vertical histogram===\n");
      plotHistV(bin, freq, N, S, B);
}
```

Fig. 3. Printing histograms from the .txt file.

The program then checks if a second file is specified, otherwise it terminates. It starts by passing the second argument (`argv[2]`) to a `readTextFile` function along with pointers to the `word` array and variables `W`, and `L`. The function reads the file and stores and stores all the strings in the file into the `word` array. The `sortWordLen` function then processes the strings in the `word` array and stores the word lengths in the `bin` array, as well as the frequency of the word lengths in the `freq` array. Two histograms are then plotted the same way as before.

3 Sub-functions

The sub-functions used in this program is declared in the `histogram.h` file and defined in the `histogram.c` file. The header file is then included in the `main.c` file. This helps ensure modularity of the code. The sub-functions are categorised into 3 sections: Operational Functions, File Read Functions, and Histogram Plotting Functions.

3.1 Operational Functions

These set of functions perform operations on the data passed to them as arguments.

3.1.1 `findLargestF()`

```
float findLargestF(float *arr, int N){
    int i;

    float max = arr[0];

    for (i=1;i<N;i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }

    return (max);
}
```

Fig. 4. Function to find the largest float in an array.

This function finds the maximum float in an array. It takes in two arguments, an array of floats, and the total number of floats in the array. The largest float is then found by comparing each element of the array with the current maximum value initialised as 0, and if a larger number is found, it replaces the current maximum value with the new number. The function returns this maximum value as a float to the calling environment.

3.1.2 `findLongestS()`

```
int findLongestS(char **arr, int W){
    int i, l;

    int max = 0;

    for(i=0;i<W;i++){
        l = strlen(arr[i]);

        if(l > max){
            max = l;
        }
    }

    return (max);
}
```

Fig. 5. Function to find the longest string in an array.

This function searches for the longest string in an array. It takes in two arguments, an array of strings, and the total number of strings in the array. The longest string is found by comparing the length of each string in the array with the current maximum value initialised as 0, and if a longer string is found, it replaces the current maximum value with the new string length. The function then returns the maximum value as an integer to the calling environment.

3.1.3 `clearHist()`

```
void clearHist(float *bin, float *freq, int *N){
    memset(bin, 0, 100*sizeof(float));
    memset(freq, 0, 100*sizeof(float));
    *N = 0;
}
```

Fig. 6. Function to reset the histogram data.

This simple function resets the arrays and variables used to plot the histogram. It uses `memset` to set all elements of the array to 0, so that it can accept new data. The number of bins, `N`, is also reset.

3.1.4 `sortWordLen()`

```
void sortWordLen(char **word, float *bin, float *freq, int W, int *N){
    int i, j, l;

    *N = findLongestS(word, W);

    for(j=0;j<*N;j++){
        bin[j] = j+1;
    }

    for(i=0;i<W;i++){
        l = strlen(word[i]);
        freq[l-1]++;
        free(word[i]);
    }
}
```

Fig. 7. Function to sort strings according to their length and count the frequency.

The function uses `findLongestS` to find the longest string in the `word` array, and initialises it to the number of bins, `N`. A for loop is then used to initialise all the bins up to the length of the longest string. Another for loop iterates through every string stored in word and counts its length using `strlen`, then increments the count for the corresponding `freq` array index. The word array is then freed as its memory was dynamically allocated in the `readTextFile` function.

3.2 File Read Functions

The following functions are used to extract data from files.

3.2.1 `readHistFile()`

```
void readHistFile(char *filename, float *bin, float *freq, int *N, int L){
    FILE *f;

    if((f=fopen(filename,"r")) != NULL){
        int i = 0;

        printf("\nReading .hist file...\n");

        while((fscanf(f, "%f %f\n", &bin[i], &freq[i])) != EOF){
            if(i < L){
                i++;
            }
            else{
                printf("\nArray limit of %d reached ..., L);
                printf("Histogram will be limited to the first %d bins ...", L);
                break;
            }
        }

        *N = i;
    }
    else{
        printf("\nFile not found! :(\n");
    }
}
```

Fig. 8. Function to extract data from a .hist file.

This function accepts 5 arguments: the filename specified by the user, the `freq` and `bin` array, the bin counter, `N`, and the size of the array, `L`. It first checks to see if the file specified is valid and returns an error otherwise. The function then scans the file following a pattern of 2 floats per line using `fscanf` until it reaches the End of File (EOF) character. At the same time, it increments a counter by one for every line read. This gives the number of bin in the `.hist` file, and is passed to `N`. Additionally, it checks to see if the maximum size of the array is reached as to make sure that the file does not overflow the array.

3.2.2 `readTextFile()`

```
void readTextFile(char *filename, char **word, int *W, int L){
    FILE *f;

    char str[50];

    if((f=fopen(filename,"r")) != NULL){
        int i = 0;

        printf("\nReading .txt file...\n");

        while((fscanf(f, "%49[a-zA-Z0-9'] %*[^a-zA-Z0-9']", str)) != EOF){
            if(i<L){
                word[i] = (char*)malloc(strlen(str) + 1);
                strcpy(&*word[i], str);
                i++;
            }
            else{
                printf("\nWord array limit of %d reached ...", L);
                printf("Histogram will be limited to the first %d words ...", L);
                break;
            }
        }

        *W = i;
    }
    else{
        printf("\nFile not found! :(\n");
    }
}
```

Fig. 9. Function to extract data from a .txt file.

The function uses a buffer array to first store the string before passing it to the word array. Doing this allows memory for the word array to be dynamically allocated using `calloc` based on the string length so it is more memory efficient. While scanning, a regular expression is used to only scan for lowercase letters, uppercase letters, numbers, and apostrophes, ignoring all other punctuation. It is limited to scan for only 49 characters to prevent the `str` array from overflowing. Extra measures are also taken to ensure that a text file with more words than allocated for does not overflow the word array. The total number of words in the file, `W`, is also counted.

3.3 Histogram Plotting Functions

These functions are used to plot histograms.

3.3.1 `printHist()`

```
void printHist(float *bin, float *freq, int N){
    int i;

    printf("\n|===bin===|===freq===|\n");

    for(i=0;i<N;i++){
        printf("|   %6.2f|    %6.2f|\n", bin[i], freq[i]);
    }

    printf("|=========|==========|\n");
}
```

Fig. 10. Function to print raw data from .hist file.

The function iterates through every element in the `bin` and `freq` array and prints them out in a table. Some extra formatting is used to make the numbers easier to see.

3.3.2 `plotHistH()`

```
void plotHistH(float *bin, float *freq, int N, int S, char B){
    int i,j;
    int l[N];

    float m = findLargestF(freq, N);

    for(i=0;i<N;i++){
        l[i] = S*freq[i]/m;
    }

    printf("|-------------|\n");
    printf("|bin    |  freq|\n");
    printf("|-------------|\n");

    for(i=0;i<N;i++){
        printf("|%-6.2f|%6.2f|", bin[i], freq[i]);

        for(j=1;j<=l[i];j++){
            printf("%c", B);
        }

        printf("\n");
    }

    printf("|-------------|\n");
}
```

Fig. 11. Function to plot a horizontal histogram.

This function starts by using the `findLargestF` function to find the mode of the histogram and stores it into variable `m`. Using this value, the ratio for the bar length of each bin, can be calculated and multiplied by the maximum bar length, `S`, to get a scaled length `l[i]`. This allows for the 'resolution' of the histogram to be changed. A for loop then iterates through every bin and prints out the data and a nested for loop prints out the histogram bar using the selected character, `B`, up to `l[i]`.

3.3.3 plotHistV()

```
void plotHistV(float *bin, float *freq, int N, int S, char B){
    int i,j;
    int l[N];

    float m = findLargestF(freq, N);

    for(i=0;i<N;i++){
        l[i] = S*freq[i]/m;
    }

    for(j=S;j>=0;j--){
        for(i=0;i<N;i++){
            if(l[i]>j){
                printf(" %c ", B);
            }
            else{
                printf("   ");
            }
        }
        printf("\n");
    }
...
```

Fig. 12. Function to plot a vertical histogram.

Same as the `plotHistH` function, the function starts by finding the mode of the histogram and ratios the frequency of all bins against the mode to get a scaled histogram bar length. To print the histogram vertically, it uses a nested for loop to iterate through every bin before going to the next line. It then compares the length of the bar, `l[i]`, against the current line number, and if it is larger, it prints the selected character, `B`. Otherwise, it prints a blank space.

4 Results

The program compiles and runs without any errors and outputs the expected results. Pictures of the output can be found in Appendix A.

4.1 Problems Encountered

During the designing and testing stage, a few problems were encountered. One of them was the scaling of the histogram as it was quite inflexible and required individually modifying the amount of characters to get a desired size, especially when the frequency is large. As such, a few functions were declared to find the mode of the histogram for the purposes of getting the ratio of the frequency against the mode, and a new variable to set the scale of the bars, `S`, was declared and passed to the functions. This allows the size of the histogram to be controlled by 1 variable.

When plotting the vertical histogram, the bin numbers at the bottom would run out of alignment whenever 2 digits were reached. An if else statement was implemented so that it prints 1 less space if there are 2 digits.

After creating the `readTextFile` function, it was also made clear that punctuation was counted as part of the word length. To solve this, a regular expression was implemented to only accept letters and numbers. However, this would split words with apostrophes as 2 different words, so the apostrophe was added as part of the regular expression.

Finally, the memory for the word array was originally statically allocated to 50 chars per string. Considering that most words take up less than 10 characters, and multiplying that by the number of strings allocated, this meant that there was a lot of unused space by each string. As such, the `calloc` function was used to dynamically allocate the memory for each string based on the length of the string, so that it is more memory efficient.

4.2 Limitations of the Program

There are a couple of limitations to this program. For one, the array sizes of the `bin`, `freq`, and `word` array are static. Because of this, there is a maximum predefined number of data allowed to be read by the program. The program also does not check for file type, so any file can be passed as arguments to the program, which may cause the

program to produce unexpected results. When printing the histogram from a `.hist` file, the data is not sorted first. Therefore, it prints out the histogram based on the sequence it was scanned in the file. Lastly, the regular expression used while scanning the text file counts the apostrophe as part of the word length. The alternative was to have words with apostrophes split into two. Ultimately, it was decided that the apostrophe should be included so that the number of words is accurate.

Appendix A

Results of the program.

```
Reading .hist file...

|===bin===|===freq===|
|     1.00|      3.14|
|     2.00|     11.11|
|     3.00|      8.80|
|     4.00|     15.20|
|     5.00|      1.17|
|     6.00|      3.52|
|     7.00|      1.00|
|     8.00|      2.98|
|     9.00|     13.29|
|    10.00|      6.30|
|=========|==========|
```

Fig. I. Printing out histogram raw data.

```
===horizontal histogram===
|-------------|
|bin    |  freq|
|-------------|
|1.00   |  3.14|ooooo
|2.00   | 11.11|ooooooooooooooooo
|3.00   |  8.80|oooooooooooooo
|4.00   | 15.20|ooooooooooooooooooooooooo
|5.00   |  1.17|o
|6.00   |  3.52|ooooo
|7.00   |  1.00|o
|8.00   |  2.98|oooo
|9.00   | 13.29|ooooooooooooooooooo
|10.00  |  6.30|oooooooooo
|-------------|
```

Fig. II. Plotting out horizontal histogram.

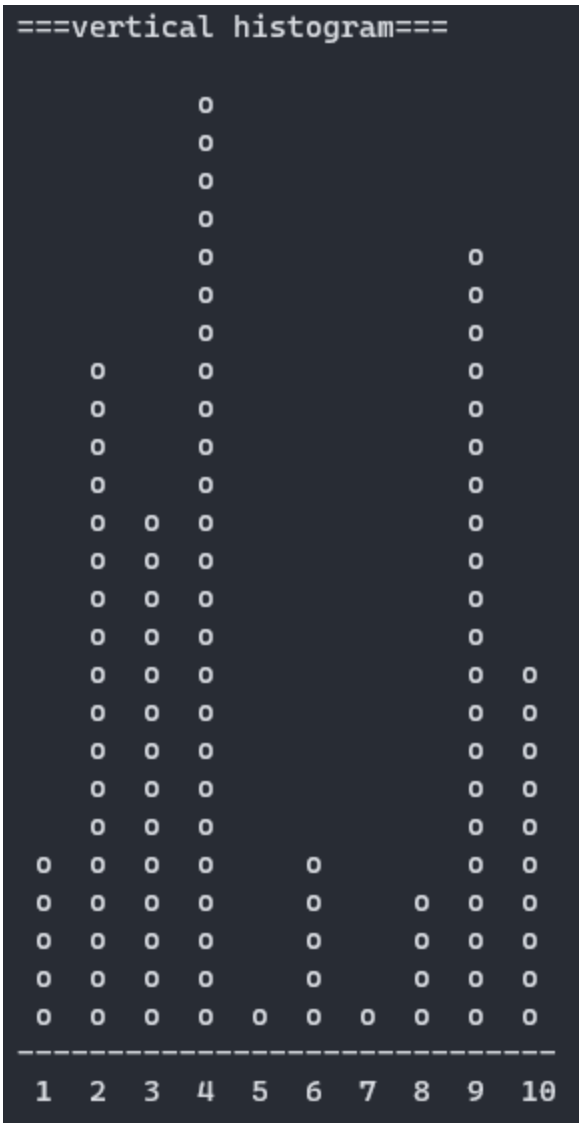

Fig. III. Plotting out vertical histogram.

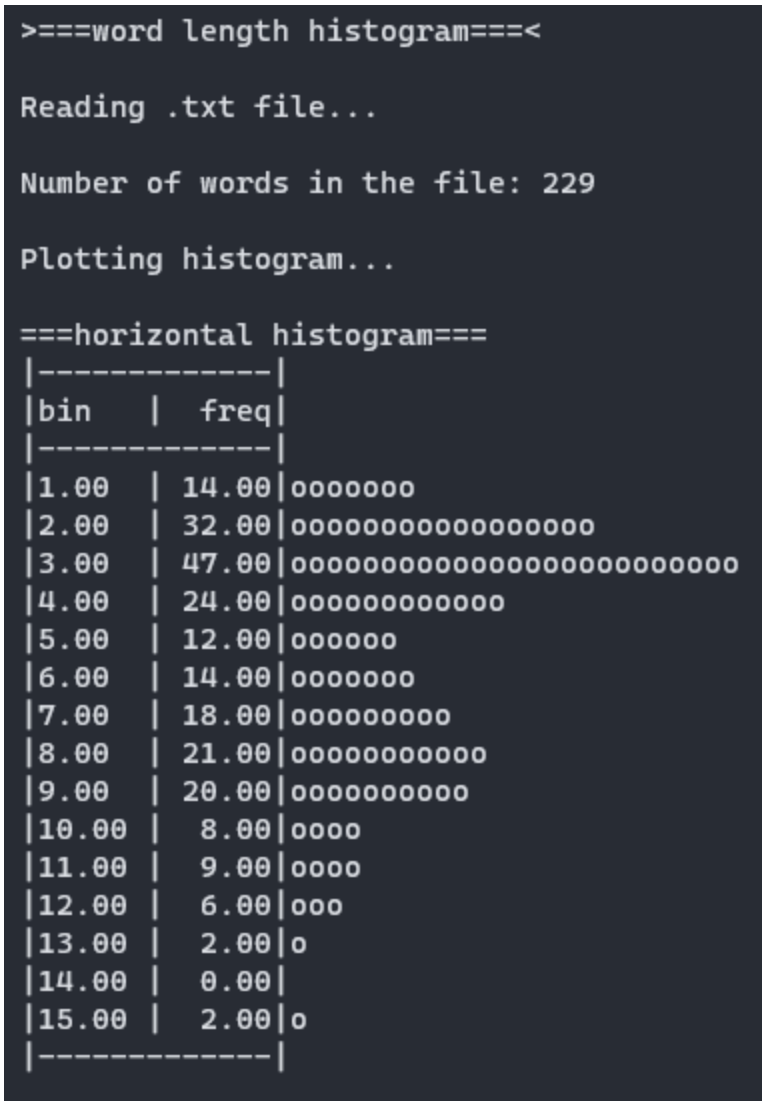

Fig. IV. Reading .txt file and plotting horizontal histogram.

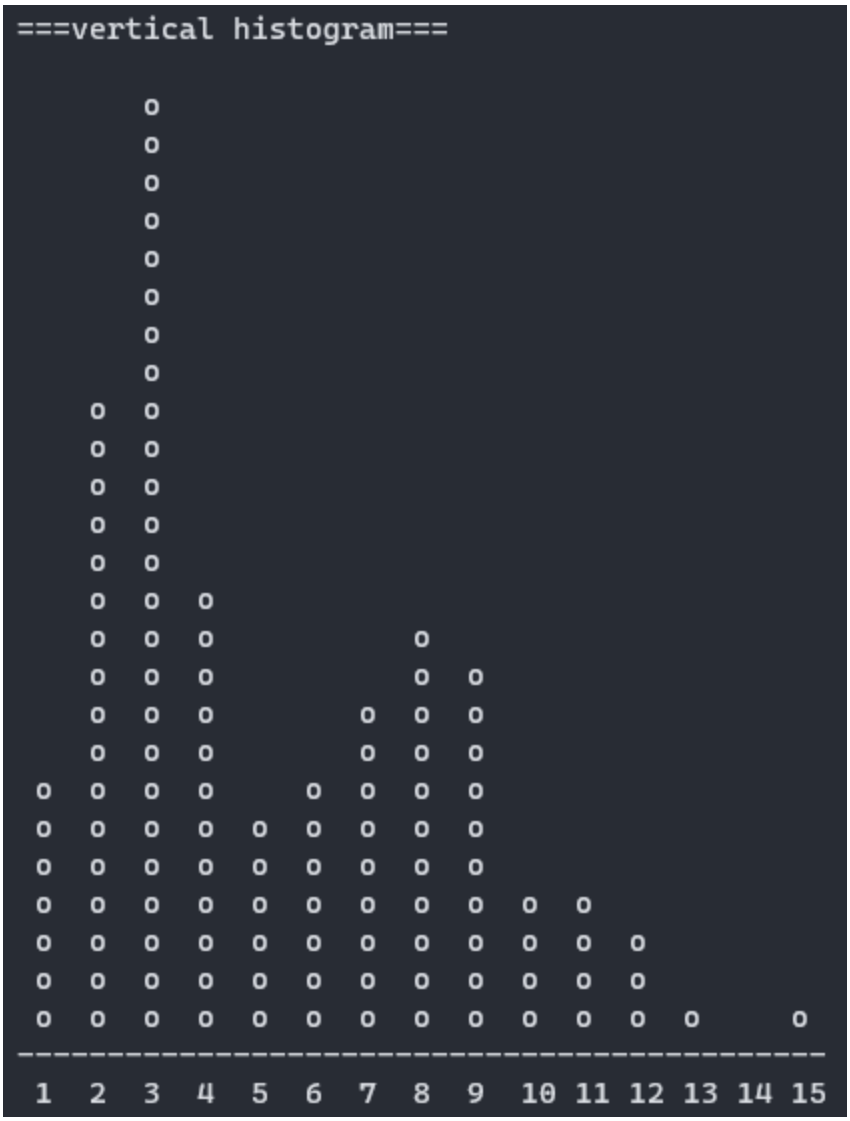

Fig. V. Plotting out vertical word length histogram.

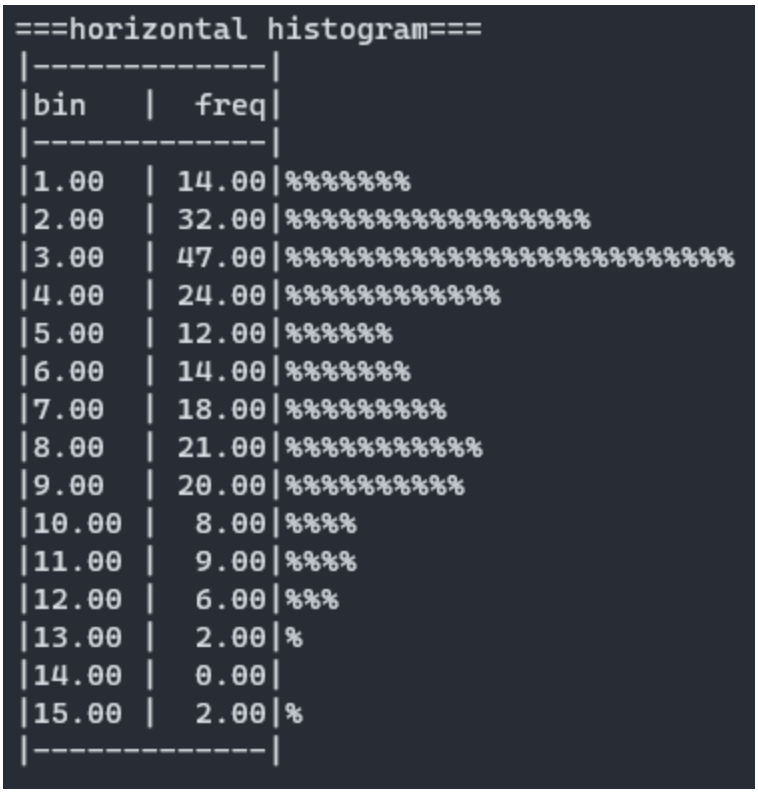

Fig. VI. Example of changing histogram bar character style.