DS4Talker

LAB #9
SECTION 5

SUBMITTED BY:

Jackson Collalti

SUBMISSION DATE:

12/06/2022

Problem

Part 1, using a given text file, output the words and characters in 5 columns rows. Part two, use the joystick to navigate the list of words. Triangle will add a word will a space after, x will remove the last item (word punctuation or letter). Square will add a word without adding a space. Bonus of adding a circle acting as a shift key to capitalize the next letter.

Analysis

The text file must be displayed in the terminal. Each button must perform the role described above. Additionally, a character must follow the joystick when trying to select a word.

Inputs: Text file and controller data

Outputs: Printing of text file, current cursor location, words, and characters selected.

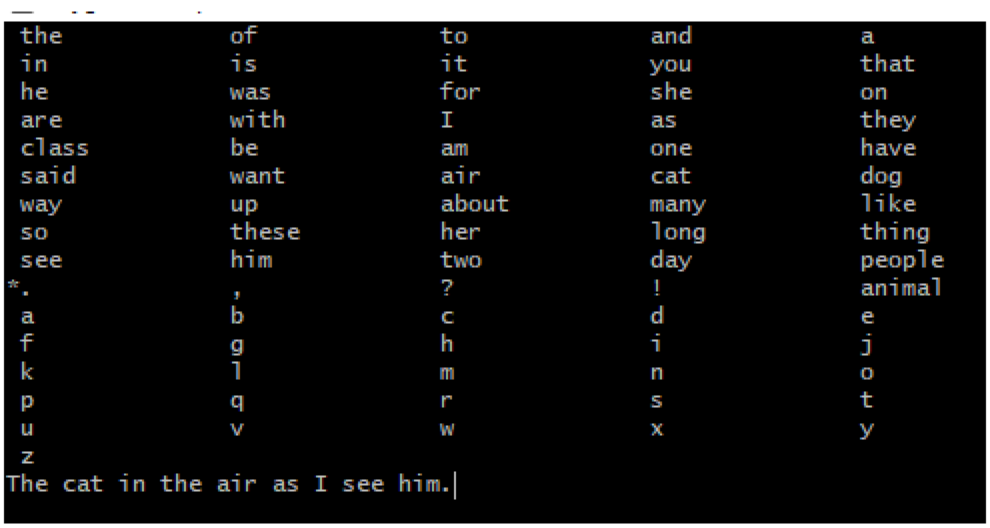
The text file and the output of the text file in the terminal are directly related. They should have the same information. To do the work with words and characters outputted depend on the movement of the joystick, and the buttons pressed are important.

Design

The first problem is to get the text file to output in the terminal. First, I need to get the file opened and get the words into my program. Second I need to get them into an array format using malloc and strcpy. Then I need to trim the unused elements by checking when the last letter in the word is. To start part two, I need to format the words into 5 columns. Knowing the max length is 15 characters for a word and 5 words max per line, I could easily get the max total characters per line. I also needed to add one character for the cursor. After that, I need to get the cursor action to work. This is similar to updating the avatar in lab 8. From that point, I need to get all the button functions to work. I first did the square button because it seemed the most relevant. After all, the triangle is a square plus a space. Once I get that, I will make the x button act as a delete.

Testing

For testing, I need to test all of the possible steps. I plan on doing step-by-step programming testing as I go. This will result in testing each feature as it's implemented in the program. Each testing part will be multiple trials testing what has been added and how it interacts with the previous steps. After the final thing is added, I plan on trying to create five to ten sentences using all the button commands. During the process of creating the sentences, I need to randomly test my delete function for the square.


```
 the           of            to            and           a
 in            is            it            you           that
 he            was           for           she           on
 are           with          I             as            they
 class         be            am            one           have
 said          want          air           cat           dog
 way           up            about         many          like
 so            these         her           long          thing
 see           him           two           day           people
*.             ,             ?             !             animal
 a             b             c             d             e
 f             g             h             i             j
 k             l             m             n             o
 p             q             r             s             t
 u             v             w             x             y
 z
The cat in the air as I see him.|
```

Comments

This was definitely the hardest lab, as it should be. I got really confused very early because I got nothing printed after multiple attempts. After about an hour or two of work, I stopped for the day. When I came back and opened my lab 9 file, I saw that the text file had 0 KB's in size. When I opened the file, it was blank and had no words. I re-downloaded the text file and ran my program, and it worked. This was painful, and a great example of a problem occurring that isn't programming related. In this lab, I learned a lot about arrays and strings. I improved my ability to develop code slowly and test it as I worked.

Lab Report Questions

1. Describe your process for reading every word from a file. What were some of the struggles you encountered?

As mentioned above in the comments, I originally, when testing, had no words in the text file, which meant nothing was printed to the screen. My approach was to set up the read words function to open the file and scan the words. As words were being scanned, I would set up a buffer. The scanning would run in a while loop that detected when no more words were left. Once I get the word scanned, I will run my trim function and use malloc for the length of the word + 1 (for the null). Then I would use strcpy to get the word into the array and repeat. When developing my read words function, I had to make sure my buffer was acting correctly, and my trim function was correct. The Hardest part was definitely figuring out how to use the buffer and malloc

2. Describe how you keep track of the word selected on the screen and how this interfaces with the DualShock 4. Is the interface reasonable?

I made it so that the row is an integer division word# / 5. This would mean word 3 ($3/5 = 0$), and 0 would represent the first line of words. To get the x coordinate of a word, I did (word# % 5) to get the word it is on the line (7%5=2nd word on line) and multiplied it by 15 because the max characters for a word are 15. Then I added 1 because I want the cursor to appear to the left of the first character of the word. Meaning I need a single additional space. For the joystick, the design is what would be expected. Joystick Down = Cursor down, Joystick Up= Cursor Up, Joystick Left = Cursor Left, Joystick Right = Cursor Right. The cursor display was the same as the avatar in lab 8. Update it to a new location, add the symbol (in my case *) and delete the symbol from the previous location. The way it worked is if up or down happened, the word # would either go up or down 5 (curAt -= 5 or curAt +=5). This would work because a full line is 5 words, so 5 words forward are the word below the current word. Horizontal was similar but only in increments of 1's (curAt -= 1 or curAt +=1). The interface is reasonable but far from perfect. The joystick and button commands make sense. The biggest issue is the problem of very large lists and having large scroll lengths. In a real application, the addition of alphabetization and the addition of larger leaps would be necessary. For actual daily use, I think the speed of selecting words and the time it takes to make a full sentence is a big problem. This is more opinion based but I think the buttons on the left side of the controller are better than the joystick for movement.

```
// Lab 9 DS4Talker Skeleton Code

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <ncurses.h>
#define MAXWORDS 100
#define WORDLEN 11
#define SENTENCE_LEN 81
#define DEBUG 0   // set to 0 to disable debug output

// reads words from the file
// into wl and trims the whitespace off of the end of each word
// DO NOT MODIFY THIS Prototype
int readWords(char* wl[MAXWORDS], char* filename);

//modifies s to trim white space off the right side
// DO NOT MODIFY THIS Prototype
void trimws(char* s);

int drawWords(char* wl[MAXWORDS], int numWords);
int addSent(char sentence[SENTENCE_LEN], char add[WORDLEN]);

int main(int argc, char* argv[])
{
        char* wordlist[MAXWORDS];
        int wordCount;
        int i;

        wordCount = readWords(wordlist, argv[1]);

        if (DEBUG)
        {
                printf("Read %d words from %s \n", wordCount, argv[1]);
                // add code to print the words to the screen here for part 1
                for (i = 0; i < wordCount; i++)
                {
                        printf("%s\n", wordlist[i]);
                }
        }

        // most of your code for part 2 goes here. Don't forget to include the ncurses library

        int t, tri, cir, x, sq;
        int rb, lb, bb;
        int lx, ly, rx, ry;

        initscr();
        refresh();

        char sent[81] = "";
        int capNext = 0;
        int curAt = 0;

        int changed = 0;

        mvaddch(0, 0, '*');
        int row = drawWords(wordlist, wordCount);
        int lastMov = 0;

        int undo[80] = { 0 };
        int undoAt = 0;

        /*
```

```
	The position of the cursor is determined by the spot in the array. Moving the joystick 
sideways moves
	the position forward or backwards 1, if there is an element in the array in that direction. 
Moving
	the joystick up or down moves the spot forward or bakwards 5 since each line contains 5 
elements.
	this is a logical way to move the cursor since it appears to move the cursor in the 
direction of the joystick.
	*/

	do
	{
		scanf("%d, %d,%d,%d,%d, %d,%d,%d,%d,%d,%d,%d,%d, %d, %d, %d, %d", &t, 
&tri, &cir, &x, &sq, &rb, &lb, &bb, &bb, &bb, &bb, &bb, &bb, &lx, &ly, &rx, &ry);

		int oldAt = curAt;

		if (0 == changed)
		{
			changed = 1;

			if (1 == tri)
			{
				int added = addSent(sent, " ");
				if (added > 0)
				{
					undo[undoAt] = added;
					undoAt++;
				}
			}
			else if (1 == cir)
			{
				capNext = 1;
			}
			else if (1 == x)
			{
				if (undoAt > 0)
				{
					sent[strlen(sent) - undo[undoAt - 1]] = '\0';
					undoAt--;
				}
			}
			else if (1 == sq)
			{
        if (1 == capNext)
        {
            wordlist[curAt][0] = toupper(wordlist[curAt][0]);
            int added = addSent(sent, wordlist[curAt]);
            if (added > 0)
            {
                undo[undoAt] = added;
                undoAt++;
            }
            wordlist[curAt][0] = tolower(wordlist[curAt][0]);
            capNext = 0;
        }
        else
        {
            int added = addSent(sent, wordlist[curAt]);
            if (added > 0)
            {
                undo[undoAt] = added;
                undoAt++;
            }
            undoAt++;
        }
    }
    else if (1 == lb)
    {
        sent[0] = '\0';
    }
    else
    {
        changed = 0;
    }
}
else if (0 == tri && 0 == sq && 0 == x && 0 == lb)
{
    changed = 0;
}

if (t - lastMov > 500)
{
    lastMov = t;
    if (lx > 100)
    {
        if (curAt < wordCount - 1)
        {
```

```
                                curAt++;
                                changed = 1;
                        }
                }
                else if (lx < -100)
                {
                        if (curAt > 0)
                        {
                                curAt--;
                                changed = 1;
                        }
                }
                else if (ly > 100)
                {
                        if (curAt < wordCount - 6)
                        {
                                curAt += 5;
                                changed = 1;
                        }
                }
                else if (ly < -100)
                {
                        if (curAt > 4)
                        {
                                curAt -= 5;
                                changed = 1;
                        }
                }
                else
                {
                        t = 0;
                }
        }
        else if (lx > -100 && lx < 100 && ly > -100 && ly < 100)
        {
                t = 0;
        }

        if (1 == changed)
        {
                mvprintw(row, 0, "%s", "
");
                mvprintw(row, 0, "%s", sent);
                if (oldAt != curAt)
                {
                    mvaddch(oldAt / 5, (oldAt % 5) * 15, ' ');
                    mvaddch(curAt / 5, (curAt % 5) * 15, '*');
                }

                refresh();
            }
    } while (1);

    return 0;
}

int addSent(char sentence[SENTENCE_LEN], char add[WORDLEN])
{
    if (strlen(sentence) + strlen(add) < SENTENCE_LEN - 1)
    {
        strcat(sentence, add);
        return strlen(add);
    }

    return 0;
}

int drawWords(char* wl[MAXWORDS], int numWords)
{
    int at = 0;
    while (at < numWords)
    {
        mvprintw(at / 5, (at % 5) * 15 + 1, "%s", wl[at]);
        at++;
    }
    refresh();

    return at / 5 + 1;
}

/*
This scans a word from the file and stores it in a buffer. Each time it reads a word, it removes any white space at the end then adds it to
the list of words. To add it to the list of words, it allocates memory for the length of the string plus 1 for the null
character. The buffer string is then copied into the memory spot allocated in the word list.
Getting use to allocating memory and how that works took a while to figure out and understand.
```

```
*/
int readWords(char* wl[MAXWORDS], char* filename)
{
	FILE* f = fopen(filename, "r");
	int at = 0;
	char buf[WORDLEN];
	while (1 == fscanf(f, "%s", buf))
	{
		trimws(buf);
		wl[at] = malloc(strlen(buf) + 1);
		strcpy(wl[at], buf);
		at++;
	}

	return at;
}

void trimws(char* s)
{
	int len = strlen(s);

	int i = len - 1;
	while (0 <= i && 1 == isspace(s[i]))
	{
		s[i] = '\0';
		i--;
	}
}
```