CPRE 1850 – DS4Talker

LAB 9
SECTION CN

SUBMITTED BY:

MAXWELL MILLER

12/12/2025

Describe your process for reading every word from a file and what steps/commands that would be needed for each word

The **Read Words** function first opens a file stream with the provided filename. Then the function starts a while-true loop. Each iteration of the while-true loop, it first allocates the memory for that index with the maximum size of a word. Then it reads the line from the file stream, however, if the **fgets** function returns false, we know that there are no more lines to read, so the loop is broken. Otherwise, we then trim the white space off the end of the line using the **Trim White Space** function. For each line read, we also increase a counter variable to then return at the end of the function.

The biggest struggle that I ran into was figuring out exactly how the file stream works in C, and how exactly lines are read. Once I figured out how to detect the end of the file, and how to open the file in the first place, however, it was pretty smooth sailing.

The **Trim White Space** function, that is a helper function for **Read Words**, is even more simple. Essentially it loops through each character in the string until it reaches a space or a new-line character and replaces it with the null character and then exits.

Describe how you would keep track of the word selected out of the wordlist when on the screen in the columns

The way the program keeps track of the selected word is by having a selection index that is increased or decreased by the DualShock controller. This selection index – called **sel** in the program – is confined by the valid indices of the **Word List** array. The way the DualShock controller interfaces with this is by using either of the joystick inputs, pushing left or right.

Unfortunately, however, the speed that the DualShock updates is way too fast for this purpose, so I implemented a debounce time and an aggressive tolerance to ensure the selection change was really what the user intended to do.

Describe how to "delete" words from the sentence if you know the lengths of the individual items in the sentence

In theory it is as simple as setting the **[strlen(output) – delete_length]** index to be the null character (**'\0'**). However, my implementation uses a very different method, which is described in the comments section on the next page.

Is the joystick easier to program and to use for movement compared to using the gyro in Lab 8? Why?

The joystick is way easier to use, in my opinion, as it outputs an integer and doesn't have nearly as much noise in the data. This makes it easier to set thresholds and have consistent outputs, as we don't have to deal with floating point numbers like we do for gyro data.

Comments

I have included my code implementation despite it not being required.

While the way I implemented it is **by far** not the most efficient way to accomplish the goal, I am quite proud I managed to make it work the way I did. I essentially used an array of that contain each word in the output string, and then when it was output, I reassembled the string. This resulted in using significantly more memory and processing time, but it did work.

Ultimately, I chose to do it this way because I thought it would be an interesting solution to the problem.

Code implementation

```c
/*------------------------------------------------------------------------------
-                       CprE 1850 Lab 09 - Part 2
------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
-                                Includes
------------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <ncurses.h>
// #include <ncurses/ncurses.h>

/*------------------------------------------------------------------------------
-                                Defines
------------------------------------------------------------------------------*/
#define MAXWORDS 100
#define WORDLEN 11

#define WORDSPACING 15
#define MAXCOLS 5
#define LEFT_PADDING 1      // Minimum 1 for selection cursor

#define MAXOUTPUTLEN 80

#define DEBUG 0   // set to 0 to disable debug output

#define JOYSTICK_THRESHOLD 100
#define DEBOUNCE_TIME 200
```

Continued on the next page.

```c
typedef struct {
    int tri;    // Triangle
    int cir;    // Circle
    int x;      // Cross
    int sqr;    // Square

    int js1;    // Joy Stick 1
    int js2;    // Joy Stick 2

    int opt;    // Options
    int share;  // Share

    int tg1;    // Trigger 1
    int tg2;    // Trigger 2
    int bp1;    // Bumper 1
    int bp2;    // Bumper 2
} ButtonState;

typedef struct {
    int x;
    int y;
    int button;
} JoystickState;


/*-----------------------------------------------------------------------------
-                              Prototypes
-----------------------------------------------------------------------------*/
int readWords(char* wl[MAXWORDS], char* filename);
void trimws(char* s) ;

void draw_words(char* wordlist[MAXWORDS], int wordCount);
void printOutput(char* outputArr[], int length);
void draw_character(int x, int y, char use);
```

Continued on the next page.

```
/*-------------------------------------------------------------------------
-                            Implementation
-------------------------------------------------------------------------*/
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]);
        }
        // If we're debugging, just exit after printing the words
        return 0;
    }

    char* outputArr[MAXOUTPUTLEN];
    int outI = -1;      // Output index in outputArr[]
    int outLen = 0;     // Current length of outputArr string

    // Current selection index
    int sel = 0;
    int newSel = 0;
    int selR = 0, selC = 0;     // Selection row and column helper variables

    char* selWord;

    // Capitalize next word flag
    int capNext = 0;

    // Variables for reading ds4rd.exe input
    int t = 0;
    int last_t = 0;
    ButtonState bState;
    JoystickState jState1;
    JoystickState jState2;

    initscr();
    refresh();
```

Continued on the next page.

```
    draw_words(wordlist, wordCount);

    int numRows = wordCount / MAXCOLS;
    if (wordCount % MAXCOLS != 0) {
        numRows += 1;
    }

    draw_character(0, LEFT_PADDING - 1, '>');

    // Main loop
    while (1) {
        // Update selection index
        newSel = -1;
        selR = sel / MAXCOLS;
        selC = sel % MAXCOLS;


        // --------------------- Read ds4rd.exe Input ---------------------

        // ./ds4rd.exe -d 054c:05c4 -D DS4_BT -t -b -j -bt
        scanf(
            "%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d",
            &t,
            &bState.tri, &bState.cir, &bState.x, &bState.sqr,
            &bState.js1, &bState.js2, &bState.opt, &bState.share,
            &bState.tg1, &bState.tg2, &bState.bp1, &bState.bp2,
            &jState1.x, &jState1.y, &jState2.x, &jState2.y
        );

        if (t - last_t < DEBOUNCE_TIME) {
            continue;
        }
```

Continued on the next page.

```
// -------------------- Joystick Handling --------------------

// Allow user to move selection cursor with joystick in the x direction
// and y direction.  Wrap around when reaching the edges of the screen.
// If the joystick is moved right, move selection right
if (jState1.x > JOYSTICK_THRESHOLD && sel < wordCount - 1) {
    newSel = sel + 1;
}
// If the joystick is moved left, move selection left
else if (jState1.x < -JOYSTICK_THRESHOLD && sel > 0) {
    newSel = sel - 1;
}

// If joystick 2 is moved right, move selection right
else if (jState2.x > JOYSTICK_THRESHOLD && sel < wordCount - 1) {
    newSel = sel + 1;
}
// If joystick 2 is moved left, move selection left
else if (jState2.x < -JOYSTICK_THRESHOLD && sel > 0) {
    newSel = sel - 1;
}

// Otherwise if joystick 1 or 2 is moved down, clear output
else if (jState1.y > JOYSTICK_THRESHOLD || jState2.y > JOYSTICK_THRESHOLD) {
    last_t = t;
    outI = -1;
    outLen = 0;
}

if (newSel != -1) {
    last_t = t;
    // Erase previous selection cursor
    draw_character(selC * WORDSPACING, LEFT_PADDING + selR - 1, ' ');

    sel = newSel;
    selR = sel / MAXCOLS;
    selC = sel % MAXCOLS;
    // Display the selection cursor
    draw_character(selC * WORDSPACING, LEFT_PADDING + selR - 1, '>');
}
```

Continued on the next page.

```
// -------------------- Button Handling --------------------

// selWord points to the currently selected word
selWord = wordlist[sel];

// When triangle button is pressed,
// add a space along with the currently selected word to the outputArr
if (bState.tri) {
    last_t = t;
    // Make sure we don't overflow outputArr,
    // or exceed MAXOUTPUTLEN when the output string is built in printOutput()
    // +1 for space
    if (outLen + strlen(selWord) + 1 < MAXOUTPUTLEN) {
        char* tempWord = (char*)malloc(sizeof(char) * MAXOUTPUTLEN);

        strcpy(tempWord, selWord);  // Copy selected word to tempWord

        if (capNext) {

            // Capitalize the first letter of selWord
            tempWord[0] = toupper(tempWord[0]);
            capNext = 0;    // Reset capitalize flag
        }

        char *new_str = (char *)malloc(sizeof(char) * (WORDLEN + 2));

        new_str[0] = ' '; // Place the space at the beginning

        // Copy the original string after the space
        strcpy(new_str + 1, tempWord);

        outputArr[++outI] = new_str;

        // Increment output length
        outLen += strlen(tempWord); // +1 for space
    }
}
```

Continued on the next page.

```
        // When square button is pressed,
        // add the currently selected word to the outputArr string
        if (bState.sqr) {
            last_t = t;
            // Make sure we don't overflow outputArr,
            // or exceed MAXOUTPUTLEN when the output string is built in printOutput()
            if (outLen + strlen(selWord) < MAXOUTPUTLEN) {
                char* tempWord = (char*)malloc(sizeof(char) * MAXOUTPUTLEN);

                strcpy(tempWord, selWord);  // Copy selected word to tempWord

                if (capNext) {
                    // Capitalize the first letter of selWord
                    tempWord[0] = toupper(selWord[0]);
                    capNext = 0;    // Reset capitalize flag
                }

                outputArr[++outI] = tempWord;

                // Increment output length
                outLen += strlen(tempWord);
            }
        }

        // When X button is pressed,
        // remove the last word added to outputArr
        if (bState.x) {
            last_t = t;
            // Make sure there is *actually* something to remove
            if (outI >= 0) {
                outLen -= strlen(outputArr[outI]);
                outI--;
            }
        }

        // When circle button is pressed,
        // capitalize the next word added to outputArr
        if (bState.cir) {
            last_t = t;
            capNext = 1;
        }
```

Continued on the next page.

```
        // ---------------------- Output Handling ---------------------

        // Print the outputArr string to the screen
        printOutput(outputArr, outI + 1);

        refresh();
    }

    endwin();

    return 0;
}

// reads words from the file
// into wl and trims the whitespace off of the end of each word
// Students are REQUIRED to use fgets() to retreive each line from the file.
int readWords(char* w1[MAXWORDS], char* filename) {
    int i = 0;

    FILE *file_stream = fopen(filename, "r");

    while (1) {
        w1[i] = (char*)malloc(sizeof(char) * (WORDLEN + 1));

        if (!fgets(w1[i], WORDLEN, file_stream)) {
            break;
        }
        trimws(w1[i]);
        i++;
    }

    return i;
}

// modifies s to trim white space off the right side
void trimws(char* s) {
    int i;

    for (i = 0; i < WORDLEN; i++) {
        if (s[i] == '\n' || s[i] == ' ') {
            s[i] = '\0';
            break;
        }
    }
}
```

Continued on the next page.

```c
void draw_words(char* wordlist[MAXWORDS], int wordCount) {
    int row, col;
    int i;

    for (i = 0; i < wordCount; i++) {
        row = i / MAXCOLS;
        col = i % MAXCOLS;

        mvprintw(row, (col * WORDSPACING) + LEFT_PADDING, "%s", wordlist[i]);
    }

    refresh();
}

void printOutput(char* outputArr[], int arrLength) {
    // char outputStr[MAXOUTPUTLEN + 1] = "";
    char *outputStr = (char *)malloc(MAXOUTPUTLEN + 1);

    memset(outputStr, ' ', MAXOUTPUTLEN); // Fill the allocated memory with spaces
    outputStr[MAXOUTPUTLEN] = '\0';        // Null-terminate the string

    int k = 0;

    // Build output string
    for (int i = 0; i < arrLength; i++) {
        for (int j = 0; j < strlen(outputArr[i]) - 1; j++) {
            outputStr[k++] = outputArr[i][j];

        }
    }

    // To prevent leftover data from previous longer outputs,
    // we use memset to change everything to blanks after the last word.
    // int outputStrLen = strlen(outputStr);
    // memset(outputStr, ' ', MAXOUTPUTLEN - outputStrLen);
    // outputStr[MAXOUTPUTLEN - 1] = '\0';  // Null terminate

    // Output string to screen
    mvprintw(MAXWORDS / MAXCOLS, 0, "Output: %s", outputStr);
    refresh();

    free(outputStr);
}
```

Continued on the next page.

```c
// PRE: 0 < x < COLS, 0 < y < ROWS, 0 < use < 255
// POST: Draws character use to the screen and position x,y
void draw_character(int x, int y, char use)
{
    mvaddch(y,x,use);
    refresh();
}
```