DS4 TALKER LAB REPORT

LAB #9

SECTION M

SUBMITTED BY:

SHOUNAK LAHIRI

SUBMISSION DATE:

12/7/2018

Part 1:

Problem

The goal for both parts of this lab is to create a speech synthesizer which would allow people with speaking disabilities or limited mobility to communicate more easily. For part 1 of the lab, the problem consists of reading in a list of words from a text file, removing new line characters, and then printing them out in list form to make sure that the program is working correctly. Additionally, the program must keep track of the number of words that are in the list.

Analysis

When the program is run, the name of the text file must be typed into the command line. As the program runs it needs to keep track of the number of words that are being read line. When reading in each word, the program should eliminate the end of line character, the problem may be that the null character which is on each string needs to be replaced/added back into the string after removing the end of line character.

Design

This part of the lab can be easily broken down into smaller parts

1. Read in the words
2. Trim the new line characters

To read in the words from the text file, the user must put the name of the text file in the command line when they run the program. The skeleton code that was provided for this lab included a function prototype called readWords who received a char array (string) and a file name. The first thing to do in the function was to initialize a char array which would house each word, the lab manual specified that none of the words would be greater than 10 characters long, therefore the size of the array was 10. The next step was to open the file that the user specified, which was achieved by using an fopen statement. Then, in a while loop that checked that the program had not run into the end of the file, the program got each line of text file and temporarily stored the line in a char array of length 10. The following step was to remove the new line character from the end of the string, using the trim function (which falls under the second step of the design). After removing the new line characters, the program should use a buffer to store the string, then copy the string from the buffer into the passed array which will store all the words from the text file. To finish the function off, we added one to the counter which counted the number of words that we were reading from the text file.

To trim the words of their new line character, the program uses a trim function called trimws, which only requires the string to be sent to it in a char* form, a string. From there, a variable is declared and set to equal the length of the string minus one, to account for the null character. Then, the program enters a while loop where it checks, starting from the end of the string, if the contents of the string are whitespace (end of line character), if it is then it sets it to the null character and continues to check the string's contents until the string contains no

more whitespace characters. Continually checking the string's content ensures that the string is in fact all letters that make up a word, even if they have varying lengths.

Testing

To test that the program was working correctly, we ran the program with the provided text file, wordslist.txt, which we knew contained 76 words. After running the program a few times and changing some small error, we were able to confidently say that the program was working correctly. As always, the ultimate test to see if the program was working correctly is to demo it to the teaching assistants.

Comments

I think that the design I implemented was good because the program worked correctly and it worked in a timely manner. The biggest thing that I learned while working on this lab is that there is often a need to have a buffer when working with a large amount of incoming data, even when it does not seem necessary.

Source Code:

```
// Lab 9 DS4Talker Skeleton Code


#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXWORDS 100
#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 main(int argc, char* argv[]) {
	char* wordlist[MAXWORDS];
	int wordCount;
	int i;
	wordCount = readWords(wordlist, argv[1]) - 1;

	if (DEBUG) {
		printf("Read %d words from %s \n", wordCount, argv[1]);
			for (int 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

    return 0;
      }

//Reads in words from text file
int readWords(char* wl[MAXWORDS], char* filename)
{
      int num = 0; //Number of words
      char word[10]; //Temporary words
      char *p;
      FILE* f = fopen(filename,"r");
      while (feof(f) != 0)
      {
            fgets(word, 10, f);
            if (p != NULL)
            {
                  trimws(word);
                  wl[num] = (char *)  malloc(strlen(word));
                  strcpy(wl[num], word);
                  num += 1;
            }
      }
      return num;
}

//Gets rid of white space characters
void trimws(char* s)
{
        int in;
        in = strlen(s) - 1;
        while ((isspace(s[in])) && (in >= 0))
            {
            s[in] = '\0';
            x -= 1;
        }
}
```

Part 2:

Problem:

In the first part of the lab we were able to get the program to read words from a text file, remove their new line characters, and keep track of the number of words we read in from the text file. Additionally, using the debug branch of an if statement, we were able to see the words that were read in and the number of words that we read in from the text file. For this part of the lab we need to alter the program to include controls for the user which will allow them to select words to place into a sentence. Additionally, the program must enable the user to add words with spaces and without, while also enabling the user to undo each change that it undergoes (each step).

Analysis

It should be noted that the order of incoming data from the joysticks is in the order of the following, left joystick: horizontal, left joystick: vertical, right joystick: horizontal, right joystick: vertical. If the horizontal joystick motion was negative, the joystick was being pushed to the left, right if the value was positive. If the vertical joystick motion was negative, the joystick was being pushed downward, positive means that the joystick was being pushed upward.

Design

The problem presented in this part of the lab can easily be broken into smaller parts

1. Get the joystick to move correctly
2. Add the word with and without spaces to the sentence
3. Remove words from the sentence

To get the program to move the cursor that was on the screen, by moving the right joystick on the Dual shock 4 controller, we needed to add variables for the incoming joystick data to the scanf statement. After setting the values form the scanf statement, we needed to use the values to control the movement of the cursor. Using only the values from the right joystick, we began the statement by testing the horizontal movement, if the horizontal value was negative we moved the cursor left by adjusting the current x coordinate by subtracting 15 and making sure that our current x coordinate was above zero, else if the value was positive and our current x coordinate was 15 less than the max (75) (to account for the movement right) then we adjusted the x coordinate by adding 15 to it. To control the vertical motion, we tested the incoming vertical data for the right joystick, if the value was negative and our current y coordinate was greater than zero we moved the cursor down by 1, else if the value was positive and the current y coordinate was less than the total number of rows minus one (to account for the movement up), we moved the cursor up by one. The number of rows that were in the program was calculated by diving the number of words in the list by 5, for the max of five columns per row. We figured the maximum size of the row by doing simple math, knowing that there could me a mx of 5 words per row and each word had a width of 15, we multiplied 5 band 15 to get 75. After completing the if

statement, we needed to set the current x and y coordinates to the variables which held the previous x and y coordinates, this allows the program to delete the cursor from the last location.

The process to add the selected word to the sentence with a space and without a space is very similar. First, a large if then else statement is constructed which tests which button on the controller is pressed. If the triangle button is pressed, then the word that is selected needs to be added to the sentence with a space appended to the end of it. To achieve this, the current coordinates on the screen need to be converted to the index of the word in the array which contains the words from the incoming text file, wordlist. To do this, the current x and y coordinates are sent to the screenToIndex function. Then, a variable which holds the length of the last word added, helpful for deleting words, gets the length of the word from the adjusted index in the wordlist array and adds one to it to account for the space. A temporary char array is constructed which will hold the word, using strcpy the word is put into the temporary array. Finally, using the mvprintw statement, then word is printed in the sentence at the bottom of the screen and adding the length of the word to the length of the sentence variable. If the square button is pressed, then the word needs to be added without a space appended to the end of it. The process for doing this is similar, the only slight change is that the length of the last word variable is just the length of the word, not plus one.

Although we did not get the program to correctly remove words, we were able to get some of the sentence to go away when the x button on the controller was pressed. To get the program to delete like we had it, we had to again get the current index from the screen x and y coordinates by calling the screenToIndex function. Then, the length of last word variable was subtracted from the length of the sentence variable, to account for deleting the word/operation. Finally, a mvprintw statement was used to replace the removed words with spaces. This did not always work because the length of the last word needed to be updated with the length of the word that was before the word that was deleted.

Testing

To test that the program was working correctly, I tried running the program and moving the joysticks to see that the cursor on the screen was moving correctly. To check that I could select words with the buttons, I pressed the buttons, checking that the correct button did the correct function and making sure it was carrying out those functions correctly. Lastly, I tried to check that the backspace was working by trying to undo some of the operations that I preformed while making sentences, but unfortunately I was not able to get this function to work correctly.

Comments

I don’t think that the design that I implemented was the best way of approaching this problem, although it does get the program to work like how it was intended to. The biggest thing that I learned from this lab is that there are a lot of ways to solve a problem, but not all of them can solve multiple problems. The way that I attempted to solve this problem worked for smaller parts of the bigger problem, but I was not able to solve the big problem. There are other ways of solving this problem that would solve both the smaller problems while also achieving a solution to the bigger problem.

Questions/Experiments

The questions are also answered in comments in my source code

1. To read every word in from the text file, we began by having the user enter the name of the file in the command line when they were trying to run the program. From there, we sent an empty char array (array of strings) and a file name to the readWords function. In the readWords function we defined a few variables, and an int which held the number of words that we read from the text file. Next, a string is declared, which will eventually, temporarily hold the word for each line. Then, a path to a file is declared which opens the specified file. A while loop is created, with its conditional stating that the loop will operate while the end of the file that was opened is not reached. Inside the while loop, the string (char*) that was declared is set to the return value of an fgets statement which will return a string, each line from the text file one at a time (one per iteration). Next an if statement is added to make sure that the string is not null, then it removes the new line character and any other white space characters from the end of the line, using the trimws function. From there a buffer is created using malloc. By using the strcpy function the string from the text file is copied into the buffer that was created by the malloc statement. Finally, we add one to the number of read words and continue iterating through the while loop until the condition is broken.
2. Towards the beginning of the program, all the words that were read in from the text file were placed into an array of strings (char* array). Each of the elements inside this array have a corresponding index. When we were calculating which x and y coordinate each word would have, we did some calculations with the index of the word, from the array of words. Notably, to get the x coordinate we did modulo division of the index by five then multiplied the result by 15. To get the y coordinate, we divide the index by 5. When we were trying to keep track of the word that was selected on the screen, we needed to go backwards… from coordinates to index. A similar calculation could be done, to get the index we divided the x coordinate by fifteen and added the result to the y coordinate multiplied by five, the result of this addition gave us the index of the word in the array which housed all the words that were read in from the text file. The Dual shock 4 controller only affected the word that was selected by having to move the cursor next to that word and when certain buttons on the controller were pressed. To keep track of the cursor when the joystick on the controller was moved around, we implemented a coordinate system. Two variable of integer type were declared to keep track of the current x and y coordinates. Each time the joystick was moved, the coordinates were updated in some way. If the joystick was moved upward the y coordinate was updated by adding one to it. If the joystick was moved downward the y coordinate was updated by subtracting one from it. If the joystick was moved to the right, the x coordinate was updated by adding one to it. If the joystick was moved to the left, the x coordinate was updated by subtracting one from it. The updated coordinates could be used to get the index of the word by following the calculations that were mentioned before. Another way that we

kept track of the word that was selected was from the interface with the Dual shock 4 controller as the buttons that were pressed can be tracked. As before mentioned, the coordinates from the screen were being tracked with each motion from the controller's joystick. When certain buttons on the controller were pressed words had to be added to the sentence on the bottom of the screen. The way that the program knew which word was selected was by making use of the coordinates and back tracking them, through the calculations mentioned before, and getting the index of the particular word in the array which housed all the words that were read in from the text file. I think that this interface is reasonable because most of the functions between the controller and the output seemed to work properly, although, there is probably a better interface out there as I was not able to get the backspacing feature to work correctly.

Source Code

```
/*
Answers to questions in lab report

1.       To read every word in from the text file, we began by having the
user enter the name of the file in the command line when they were trying
to run the program. From there, we sent an empty char array (array of
strings) and a file name to the readWords function. In the readWords
function we defined a few variables, and int which held the number of
words that we read from the text file. Next, a string is declared, which
will eventually, temporarily hold the word for each line. Then, a path to
a file is declared which opens the specified file.  A while loop is
created, with its conditional stating that the loop will operate while the
end of the file that was opened is not reached. Inside the while loop, the
string (char*) that was declared is set to the value of an fgets statement
which will return a string, each line from the text file one at a time
(one per iteration). Next an if statement is added to make sure that the
string is not null, then it removes the new line character and any other
white space characters from the end of the line, using the trimws
function. From there a buffer is created using malloc. By using the strcpy
function the string from the text file is copied into the buffer that was
created by the malloc statement. Finally, we add one to the number of read
words and continue iterating through the while loop until the condition is
broken.
2.       Towards the beginning of the program, all the words that were
read in from the text file were placed into an array of strings (char*
array). Each of the elements inside this array have a corresponding index.
When we were calculating which x and y coordinate each word would have, we
did some calculations with the index of the word ,from the array of words.
Notably, to get the x coordinate we did modulo division of the index by
```

```
five then multiplied the result by 15. To get the y coordinate, we divide
the index by 5. When we were trying to keep track of the word that was
selected on the screen, we needed to go backwards... from coordinates to
index. A similar calculation could be done, to get the index we divided
the x coordinate by fifteen and added the result to the y coordinate
multiplied by five, the result of this addition gave us the index of the
word in the array which housed all the words that were read in from the
text file. The Dual shock 4 controller only affected the word that was
selected by having to move the cursor to next to that word and when
certain buttons on the controller were pressed. To keep track of the
cursor when the joystick on the controller was moved around, we
implemented a coordinate system. Two variable of integer type were
declared to keep track of the current x and y coordinates. Each time the
joystick was moved, the coordinates were updated in some way. If the
joystick was moved upward the y coordinate was updated by adding one to
it. If the joystick was moved downward the y coordinate was updated by
subtracting one from it. If the joystick was moved to the right, the x
coordinate was updated by adding one to it. If the joystick was moved to
the left, the x coordinate was updated by subtracting one from it. The
updated coordinates could be used to get the index of the word by
following the calculations that were mentioned before. Another way that
the way we kept track of the word that was selected was interfaced with
the Dual shock 4 controller was when the buttons were pressed. As before
mentioned, the coordinates from the screen were being tracked with each
motion from the controller's joystick. When certain buttons on the
controller were pressed words had to be added to the sentence on the
bottom of the screen. The way that the program knew which word was
selected was by making use of the coordinates and back tracking them,
through the calculations mentioned before, and getting the index of the
particular word in the array which housed all the words that were read in
from the text file. I think that this interface is reasonable because most
of the functions between the controller and the output seemed to work
properly, although, there is probably a better interface out there as I
was not able to get the backspacing feature to work correctly.

*/


#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <curses.h>
#define MAXWORDS 100
#define WORDLEN 11
#define DEBUG 0
#define CURSOR '>'

//Function Prototypes
```

```
int readWords(char *wl[MAXWORDS], char* filename);
void trimws(char* s) ;
int screenX(int index);
int screenY(int index);
int screenToIndex(int y, int x);
void draw_character(int x, int y, char use);

int main(int argc, char* argv[]) {
        char* wordlist[MAXWORDS];
        int numWords;
        int i;
        numWords = readWords(wordlist, argv[1]);

        if (DEBUG)
            {
            printf("Read %d words from %s \n",numWords, argv[1]);
            for (i = 0; i< numWords; i++) {
                printf("%s,", wordlist[i]);

            }
            printf("\n");
        }



      int tempX = 0, tempY = 0, rows = 0;
      initscr();

      for(i = 0; i < numWords; i++)
      {
            tempX = screenX(i);
            tempY = screenY(i);
            mbbprintw(tempY, tempX, "%s", wordlist[i]);
      }

      rows = numWords / 5; //calcualting the number of rows

      if(numWords % rows != 0 )
      {
            rows += 1; //Adding a row if there are more than an even number
of rows (divisible by 5)
      }

      int time, jLH, jLV, jRH, jRV, bT, bC, bX, bS;
      int nowX = 0, nowY = 0, lastX = 0, lastY = 0;
      int lastLen = 0, senLen = 0;

      do
```

```
	{
		scanf("%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d",
&time, &bT, &bC, &bX, &bS, &jRD, &jLD, &option, &share, &R2, &L2, &R1, &L1,
&jLH, &jLV, &jRH, &jRV);


		draw_character(lastX, lastY, ' ');//Remove the last cursor

		//Control the movement with the joysticks
		if(((jRV > 20)||(jLV > 20)) && nowY < rows-1)
		{
			nowY++;
		}
		else if(((jRV < -20)||(jLV < -20)) && nowY>0)
		{
			nowY--;
		}
		else if(((jRH > 20)|| (jLH > 20)) && nowX < 60)
		{
			nowX+=15;
		}
		else if(((jRH < -20)||(jLH < -20)) && nowX > 0)
		{
			nowX-=15;
		}//end movement if

		//Update location for replacing in next iteration
		lastX = nowX;
		lastY = nowY;

		draw_character(nowX, nowY, CURSOR); //Draw cursor at new spot


		if(bT == 1)//Add with space
		{
			index = screenToIndex(nowY, nowX);
			lastLen = strlen(wordlist[index])+1;

			char addWord[12];
			strcpy(addWord, wordlist[index]);
			mvprintw(rows + 2, senLen," %s", addWord);
			senLen += lastLen;
		}
		else if(bS == 1) //Add without space
		{
			index = screenToIndex(nowY, nowX);
			lastLen = strlen(wordlist[index]);
			char addWord[12];
			strcpy(addWord, wordlist[index]);
```

```
                mvprintw(rows + 2, senLen,"%s", addWord);
                senLen += lastLen;
            }
            else if(bX == 1) //Undo the last operation
            {
                index = screenToIndex(nowY, nowX);
                senLen -= lastLen;
                mvprintw(rows + 2, senLen ,"              ");
            }//end button if

      }while(1);
      endwin();

}//end main

//Reads in words from text file
int readWords(char* wl[MAXWORDS], char* filename)
{
      int num = 0; //Number of words
      char word[10]; //Temporary words
      char *p;
      FILE* f = fopen(filename,"r");
      while (feof(f) != 0)
      {
            fgets(word, 10, f);
            if (p != NULL)
            {
                trimws(word);
                wl[num] = (char *)  malloc(strlen(word));
                strcpy(wl[num], word);
                num += 1;
            }
      }
      return num;
}//end readWords

//Gets rid of white space characters
void trimws(char* s)
{
        int in;
        in = strlen(s) - 1;
        while ((isspace(s[in])) && (in >= 0))
            {
            s[in] = '\0';
            x -= 1;
        }
}//end trimws

int screenY(int index)
```

```
{
	int y = index / 5;
	return y;
}

int screenX(int index)
{

	int modFive = index % 5;
	int x = modFive * 15;
	return x;
}

int screenToIndex(int y, int x)
{
	int normalX = x / 15;
	int normalY = y * 5;
	int index = normalX + normalY;
	return index;
}

void draw_character(int x, int y, char use)
{
	  mvaddch(y,x,use);
	  refresh();
}
```