Branch: master ▾ Find file Copy path

cs-35l / assignment4 / sfrob.c

prithvikannan added to lab4 and fixed comment is sfrob

ffd6308 on Oct 28

1 contributor

Raw Blame History

170 lines (150 sloc) 4.21 KB

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// implements comparison between a and b without deobfuscating
int frobcmp(char const *a, char const *b)
{
    // make sure pointers are not null
    if (a != 0 && b != 0)
    {
        // iterate through char array with pointers a and b
        while (*a != ' ' && *b != ' ')
        {
            // unfrobnicate a single byte
            char a_i = *a ^ 42;
            char b_i = *b ^ 42;

            // compare a and b and check which ends first
            if (a_i < b_i || *a == ' ')
            {
                return -1;
            }
            else if (a_i > b_i || *b == ' ')
            {
                return 1;
            }

            a++;
            b++;
        }
    }
    // a and b always equal
    return 0;
}

// custom comparator that calls frobcmp
int cmp(const void *a, const void *b)
{
    return frobcmp(*(char **)a, *(char **)b);
}

int main()
{
    char current_char;

    // allocate memory for array of strings
    char **arr = (char **)malloc(sizeof(char *));
    // allocate memory for new string
    char *temp_string = (char *)malloc(sizeof(char));
    if (arr == NULL || temp_string == NULL)
    {
        fprintf(stderr, "Memory allocation error");
        exit(1);
    }

    int string_ptr = 0;
    int char_ptr = 0;
    arr[0] = temp_string;
    bool needNewString = false;

    // increases size of string array to hold one more string
    arr = (char **)realloc(arr, (string_ptr + 1) * sizeof(char *));
    // creates the first string
    temp_string = (char *)malloc(sizeof(char));
    if (arr == NULL || temp_string == NULL)
    {
        fprintf(stderr, "Memory allocation error");
        exit(1);
    }

    // read from stdin until eof or error
    while (true)
    {
        // read text from standard input
        current_char = getchar();
        if (ferror(stdin))
        {
            fprintf(stderr, "Input read error");
            exit(1);
        }
        else if (feof(stdin))
        {
            // hit end of file, exit while loop
            break;
        }

        if (!needNewString)
        {
            temp_string = (char *)realloc(temp_string, (char_ptr + 1) * sizeof(char));
            if (temp_string == NULL)
            {
                fprintf(stderr, "Memory allocation error");
                exit(1);
            }

            // space is delimiter of new strings
            if (current_char == ' ')
            {
                needNewString = true;
            }
        }
        else // if program must create a new string
        {
            char_ptr = 0;

            // handle consecutive spaces by skipping iteration
            if (current_char == ' ' && char_ptr == 0)
            {
                continue;
            };

            string_ptr++;
            needNewString = false;

            arr = (char **)realloc(arr, (string_ptr + 1) * sizeof(char *));
            temp_string = (char *)malloc(sizeof(char));
            if (arr == NULL || temp_string == NULL)
            {
                fprintf(stderr, "Memory allocation error");
                exit(1);
            }

        }

        // add new char after adjusting pointers and allocating memory
        temp_string[char_ptr] = current_char;
        arr[string_ptr] = temp_string;
        char_ptr++;
    }

    // add trailing space if not present
    if (string_ptr != -1 && arr[string_ptr][char_ptr - 1] != ' ')
    {
        temp_string = (char *)realloc(temp_string, (char_ptr + 1) * sizeof(char));
        if (temp_string == NULL)
        {
            fprintf(stderr, "Memory allocation error");
            exit(1);
        }
        temp_string[char_ptr] = ' ';
        arr[string_ptr] = temp_string;
    }

    qsort(arr, string_ptr + 1, sizeof(char *), cmp);

    // print to stdout
    int i;
    for (i = 0; i < string_ptr + 1; i++)
    {
        int j = 0;
        while (true)
        {
            if (putchar(arr[i][j]) == EOF)
            {
                fprintf(stderr, "Printing error");
                exit(1);
            }

            // if space then move to next line
            if (arr[i][j] == ' ')
            {
                break;
            }
            j++;
        }
        free(arr[i]);
    }
    free(arr);
    exit(0);
}
```