Branch: master Find file Copy path

cs-35l / assignment5 / sfrobu.c

prithvikannan Update sfrobu.c
d8aec6b on Nov 6

1 contributor

Raw Blame History

286 lines (250 sloc) 6.18 KB

```c
#include <unistd.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <ctype.h>
#include <stdio.h>

bool isF = false;

// 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;
            char b_i;

            if (isF)
            {
                a_i = toupper((unsigned char)(*a ^ 42));
                b_i = toupper((unsigned char)(*b ^ 42));
            }
            else
            {
                a_i = *a ^ 42;
                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;
```

```c
}

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

int main(int argc, const char *argv[])
{
    switch (argc)
    {
    case 1:
        isF = false;
        break;
    case 2:
        if (argv[1][0] != '-' && argv[1][1] != 'f')
        {
            fprintf(stderr, "Invalid arguments");
            exit(1);
        }
        else
        {
            isF = true;
        }
        break;
    default:
        fprintf(stderr, "Invalid number of arguments");
        exit(1);
    }

    struct stat buf;
    fstat(0, &buf);
    size_t size;
    if (fstat(0, &buf) < 0)
    {
        fprintf(stderr, "Unable to get info");
        exit(1);
    }

    char *regFile;
    char **arr = NULL;
    int s = -1;
    bool addNewString = true;

    if (S_ISREG(buf.st_mode))
    {
        size = buf.st_size;

        regFile = (char *)malloc(sizeof(char) * (size + 1));
        if (read(0, regFile, size) < 0)
        {
            fprintf(stderr, "Unable to read");
            exit(1);
        }

        int words = 0;
        int i = 0;

        while (i < size)
        {
            // catch first char space
            if (i == 0 && regFile[i] != ' ')
            {
                words++;
            }

            if (regFile[i] == ' ')
            {
                // handle consecutive spaces by skipping iteration
                while (regFile[i] == ' ' && i < size)
                {
                    i++;
                }
                if (i < size)
                {
                    words++;
                }
            }
            i++;
        }
        regFile[size] = ' ';

        // allocate memory equal to words
        arr = (char **)malloc(sizeof(char *) * words);
        if (arr == NULL)
        {
            fprintf(stderr, "Memory allocation error");
            exit(1);
        }

        // add words to array
        for (i = 0; i < size; i++)
        {
            if (addNewString && regFile[i] != ' ')
            {
                s++;
                addNewString = false;
                arr[s] = &regFile[i];
            }
            if (!addNewString && regFile[i] == ' ')
            {
                addNewString = true;
            }
        }
    }
    else
    {
        arr = (char **)malloc(sizeof(char *));
        if (arr == NULL)
        {
            fprintf(stderr, "Memory allocation error");
            exit(1);
        }
    }

    char *temp_string;
    char input[1];
    char current_char;
    int char_ptr = 0;
    while (true)
    {

        int r = read(0, input, 1);
        if (r == 0)
        {
            break;
        }
        else if (r < 0)
        {
            fprintf(stderr, "Unable to read");
            exit(1);
        }

        current_char = input[0];

        if (!addNewString)
        {
            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 == ' ')
            {
                addNewString = 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;
            };

            s++;

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

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

    // add trailing space if not present
    if (s != -1 && arr[s][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[s] = temp_string;
    }

    // use frobcomp to sort array of strings
    qsort(arr, s + 1, sizeof(char *), cmp);

    // print to stdout
    int i = 0;
    while (i < s + 1)
    {

        int j = 0;
        while (true)
        {
            input[0] = arr[i][j];
            if (write(1, input, 1) < 0)
            {
                fprintf(stderr, "Unable to write");
                exit(1);
            }
            // if space then move to next line
            if (arr[i][j] == ' ')
            {
                break;
            }
            j++;
        }

        if (!S_ISREG(buf.st_mode))
        {
            free(arr[i]);
        }
        i++;
    }

    if (S_ISREG(buf.st_mode))
    {
        free(regFile);
    }

    free(arr);

    exit(0);
}
```