

## Day 19 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

### 1. Exercise 5-14:

Problem: Modify the sort program to handle a -r flag, which indicates sorting in reverse (decreasing) order. Be sure that -r works with -n.

Analysis: The command-line options "-n" and "-r" are both accepted by the program. The lines are sorted by number if the "-n" switch is present, enabling a numeric sort. Reverse sorting is possible if the "-r" flag is specified, which causes the lines to be arranged in reverse. In this program here I am using the all given function on the book and do those function as it is.

Additionally, in my program is implemented the reverse sort function, the lines are sorted using the reverse bubble sort method.

Test Case of this program:

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_14.exe -n -r
Enter some inputs here=>
4
3
6
9
8
1
5
^Z
The reverse sorted output is=>
9
8
6
5
4
3
1
```

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_14.exe -n -r
Enter some inputs here=>
apple
banana
cherry
date
^Z
The reverse sorted output is=>
date
cherry
banana
apple
```

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_14.exe -n
Enter your input =>
6
4
7
3
2
9
5
1
4
^Z
The sorted output is =>
1
2
3
4
4
5
6
7
9
```

Source code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXLINES 5000
char *lineptr[MAXLINES]; // Array to store input lines

int readlines(char *lineptr[], int maxlines);
void writelines(char *lineptr[], int nlines);
void reverseSort(void *lineptr[], int left, int right, int (*comp)(void *, void *));
int numcmp(const char *s1, const char *s2);

/*****
 * Function Name: main
 * Description: Entry point of the program.
 * Parameters:
 *   - argc: Number of command-line arguments.
 *   - argv: Array of strings containing the command-line arguments.
 * Returns:
 *   - 0 on successful execution, 1 on error.
 *****/
int main(int argc, char *argv[])
{
    int nlines, i, j;
    int numeric = 0; // Flag for numeric sort
    int reverse = 0; // Flag for reverse sort

    // Check command-line arguments for flags
    for (i = 1; i < argc; i++)
    {
        if (strcmp(argv[i], "-n") == 0)
            numeric = 1; // Enable numeric sort
        else if (strcmp(argv[i], "-r") == 0)
            reverse = 1; // Enable reverse sort
    }

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0)
    {
        reverseSort((void **)lineptr, 0, nlines - 1, (int (*)(void *, void *))(numeric ? numcmp : strcmp));
        if (reverse)
        {
            // Reverse the sorted lines
            for (i = 0, j = nlines - 1; i < j; i++, j--)
            {
                void *temp = lineptr[i];
                lineptr[i] = lineptr[j];
                lineptr[j] = temp;
            }
        }
        writelines(lineptr, nlines);
        return 0;
    }
    else
    {
        printf("input too big to sort\n");
        return 1;
    }
}
```

```

}

/*****
****
* Function Name: readlines
* Description: Reads lines from input and stores them in an array of pointers to strings.*
* Parameters:
* - lineptr: Array of pointers to strings where the lines will be stored.
* - maxlines: Maximum number of lines to read.
* Returns:
* - The number of lines read.

****/
int readlines(char *lineptr[], int maxlines)
{
    int nlines = 0;
    char line[MAXLINES]; // Buffer to store input line
    printf("Enter your input =>\n");
    while (fgets(line, sizeof(line), stdin) != NULL)
    {
        line[strcspn(line, "\n")] = '\0'; // Remove trailing newline character
        lineptr[nlines] = strdup(line); // Allocate memory and store line
        nlines++;
    }

    return nlines;
}

/*****
****
* Function Name: writelines
* Description: Writes the lines stored in an array of pointers to strings to the output.*
* Parameters:
* - lineptr: Array of pointers to strings containing the lines to write.
* - nlines: Number of lines to write.
* Returns:
* - None.

****/
void writelines(char *lineptr[], int nlines)
{
    int i;
    printf("The shorted output is =>\n");
    for (i = 0; i < nlines; i++)
    {
        printf("\t%s\n", lineptr[i]);
    }
}

/*****
****
* Function Name: reverseSort
* Description: Sorts an array of pointers using the reverse bubble sort algorithm.
* Parameters:
* - lineptr: Array of pointers to sort.

```

```

* - left: Starting index of the range to sort.          *
* - right: Ending index of the range to sort.          *
* - comp: Pointer to the comparison function used to determine the order of elements.*
* Returns:                                             *
* - None.                                             *

*****

***/
void reverseSort(void *v[], int left, int right, int (*comp)(void *, void *))
{
    int i, j;
    void *temp;

    for (i = left; i < right; i++)
    {
        for (j = i + 1; j <= right; j++)
        {
            if ((*comp)(v[i], v[j]) > 0)
            {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }
        }
    }
}

/*****
* Function Name: numcmp                                *
* Description: Compares two strings numerically.      *
* Parameters:                                          *
* - s1: Pointer to the first string to compare.      *
* - s2: Pointer to the second string to compare      *
* Returns:                                             *
* - -1 if s1 < s2, 0 if s1 == s2, 1 if s1 > s2.      *
*****/
int numcmp(const char *s1, const char *s2)
{
    double v1 = atof(s1);
    double v2 = atof(s2);

    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}

```

## 2. Exercise 5-15:

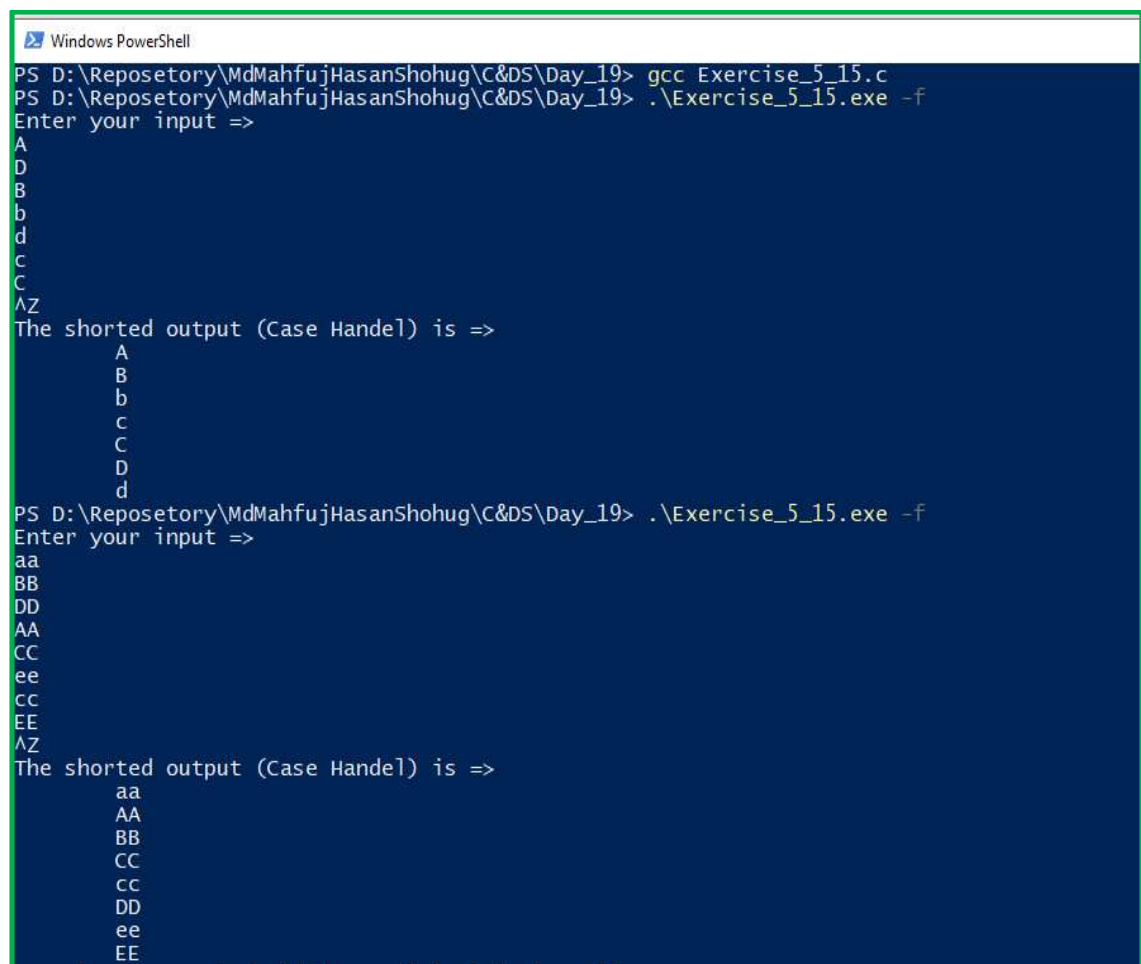
Add the option -f to fold upper and lower case together, so that case distinctions are not made during sorting; for example, a and A compare equal.

Analysis: According to the above exercise here my program modification from the Numcmp and foldcmp are two comparison functions offered by the code. Numerical comparison of two double-valued strings is performed by numcmp using atof. Foldcmp compares two strings while taking into account case. It compares the characters in the strings after converting them to lowercase using the tolower function.

The code accepts command-line arguments to modify the behavior of the program. The available flags are:

- -n: Sorts the lines numerically.
- -r: Sorts the lines in reverse order.
- -f: Sorts the lines case-insensitively.

Test case:



```
Windows PowerShell
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> gcc Exercise_5_15.c
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_15.exe -f
Enter your input =>
A
D
B
b
d
C
C
^Z
The shorted output (Case Hande1) is =>
A
B
b
c
C
D
d
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_15.exe -f
Enter your input =>
aa
BB
DD
AA
CC
ee
cc
EE
^Z
The shorted output (Case Hande1) is =>
aa
AA
BB
CC
CC
DD
ee
EE
```

```

PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_15.exe -f
Enter your input =>
Apple
cat
ball
apple
dall
Dally
^Z
The sorted output (Case Handel) is =>
    Apple
    apple
    ball
    cat
    dall
    Dally

```

Source Code:

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h> // for tolower()

#define MAXLINES 5000
char *lineptr[MAXLINES];

int readlines(char *lineptr[], int maxlines);
void writelines(char *lineptr[], int nlines);
void reverseSort(void *lineptr[], int left, int right, int (*comp)(void *, void *));
int numcmp(const char *s1, const char *s2);
int foldcmp(const char *s1, const char *s2);

/*****
 * Function Name: main
 * Description: Entry point of the program.
 * Parameters:
 *   - argc: Number of command-line arguments.
 *   - argv: Array of strings containing the command-line arguments.
 * Returns:
 *   - 0 on successful execution, 1 on error.
 *****/
int main(int argc, char *argv[])
{
    int nlines, i, j;
    int numeric = 0;
    int reverse = 0;
    int fold = 0;

    // Check command-line arguments for flags
    for (i = 1; i < argc; i++)
    {
        if (strcmp(argv[i], "-n") == 0)
            numeric = 1;
        else if (strcmp(argv[i], "-r") == 0)
            reverse = 1;
    }

```

```

        else if (strcmp(argv[i], "-f") == 0)
            fold = 1;
    }

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0)
    {
        // Choose the appropriate comparison function based on flags
        int (*comp)(void *, void *);
        if (numeric)
            comp = (int (*)(void *, void *))(numcmp);
        else if (fold)
            comp = (int (*)(void *, void *))(foldcmp);
        else
            comp = (int (*)(void *, void *))(strcmp);

        reverseSort((void **)lineptr, 0, nlines - 1, comp);
        if (reverse)
        {
            // Reverse the sorted lines
            for (i = 0, j = nlines - 1; i < j; i++, j--)
            {
                void *temp = lineptr[i];
                lineptr[i] = lineptr[j];
                lineptr[j] = temp;
            }
        }
        writelines(lineptr, nlines);
        return 0;
    }
    else
    {
        printf("input too big to sort\n");
        return 1;
    }
}

/*****
*****
* Function Name: readlines
* Description: Reads lines from input and stores them in an array of pointers to strings.
* Parameters:
*   - lineptr: Array of pointers to strings where the lines will be stored.
*   - maxlines: Maximum number of lines to read.
* Returns:
*   - The number of lines read.
*****
*****/
int readlines(char *lineptr[], int maxlines)
{
    int nlines = 0;
    char line[MAXLINES];

    printf("Enter your input =>\n");
    while (fgets(line, sizeof(line), stdin) != NULL)
    {
        line[strcspn(line, "\n")] = '\0'; // Remove trailing newline character
        lineptr[nlines] = strdup(line);
    }
}

```

```

        nlines++;
    }

    return nlines;
}

/*****
 * Function Name: writelines
 * Description: Writes the lines stored in an array of pointers to strings to the output.*
 * Parameters:
 * - lineptr: Array of pointers to strings containing the lines to write.
 * - nlines: Number of lines to write.
 * Returns:
 * - None.
 *****/
void writelines(char *lineptr[], int nlines)
{
    int i;
    printf("The shorted output (Case Handel) is =>\n");
    for (i = 0; i < nlines; i++)
    {
        printf("\t%s\n", lineptr[i]);
    }
}

/*****
 * Function Name: reverseSort
 * Description: Sorts an array of pointers using the reverse bubble sort algorithm.
 * Parameters:
 * - lineptr: Array of pointers to sort.
 * - left: Starting index of the range to sort.
 * - right: Ending index of the range to sort.
 * - comp: Pointer to the comparison function used to determine the order of elements.*
 * Returns:
 * - None.
 *****/
void reverseSort(void *v[], int left, int right, int (*comp)(void *, void *))
{
    int i, j;
    void *temp;

    for (i = left; i < right; i++)
    {
        for (j = i + 1; j <= right; j++)
        {
            if ((*comp)(v[i], v[j]) > 0)
            {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }
        }
    }
}

```



```

}

/*****
 * Function Name: numcmp
 * Description: Compares two strings numerically.
 * Parameters:
 * - s1: Pointer to the first string to compare.
 * - s2: Pointer to the second string to compare
 * Returns:
 * - -1 if s1 < s2, 0 if s1 == s2, 1 if s1 > s2.
 *****/
int numcmp(const char *s1, const char *s2)
{
    double v1 = atof(s1);
    double v2 = atof(s2);

    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}

/*****
 * Function Name: foldcmp
 * Description: Compares two strings case-insensitively.
 * Parameters:
 * - s1: First string to compare.
 * - s2: Second string to compare.
 * Returns:
 * - Negative value if s1 is less than s2.
 * - Positive value if s1 is greater than s2.
 * - 0 if s1 and s2 are equal.
 *****/
int foldcmp(const char *s1, const char *s2)
{
    while (*s1 && *s2)
    {
        if (tolower(*s1) != tolower(*s2))
            return tolower(*s1) - tolower(*s2);
        s1++;
        s2++;
    }

    return 0;
}

```

### 3. Exercise 5-17:

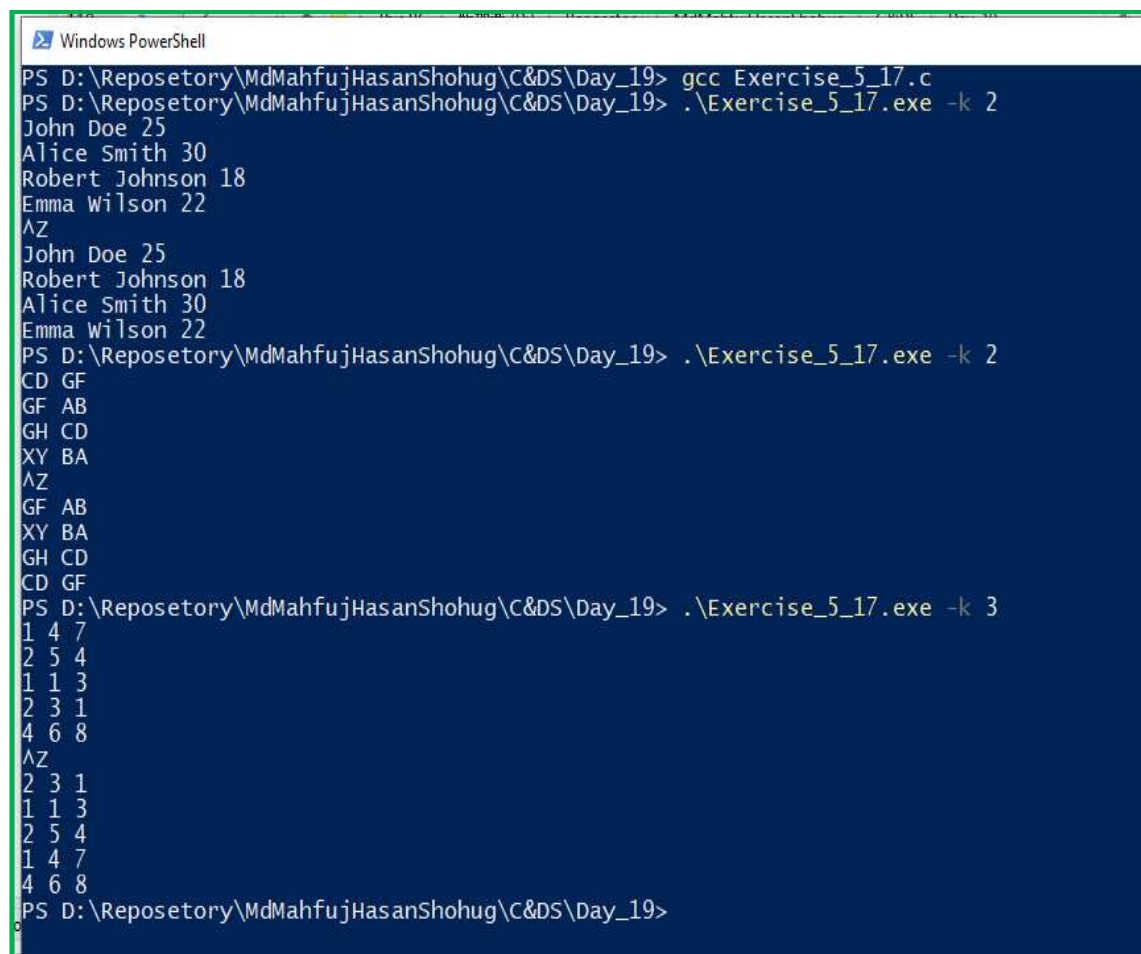
Problem: Add a field-searching capability, so sorting may be done on fields within lines, each field sorted according to an independent set of options. (The index for this book was sorted with -f for the index category and -n for the page numbers.)

Analysis:

New flag: -k (select key fields) according to the problem I change some logic on my above code.

- The code now checks for the -k flag in command-line arguments. If found, it reads the following argument as the field number (field) to consider for sorting. It allows the user to specify a key field to sort the lines based on that field.
- The -k flag is looked for in command-line arguments, and if it is, the next argument is transformed to an integer and saved in the field. With the proper argument field and the chosen comparison function, the FieldSort function is invoked. Using a loop, sorted lines are reversed if the -r flag is used.

Here is some test case:



```
Windows PowerShell
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> gcc Exercise_5_17.c
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_17.exe -k 2
John Doe 25
Alice Smith 30
Robert Johnson 18
Emma Wilson 22
^Z
John Doe 25
Robert Johnson 18
Alice Smith 30
Emma Wilson 22
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_17.exe -k 2
CD GF
GF AB
GH CD
XY BA
^Z
GF AB
XY BA
GH CD
CD GF
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19> .\Exercise_5_17.exe -k 3
1 4 7
2 5 4
1 1 3
2 3 1
4 6 8
^Z
2 3 1
1 1 3
2 5 4
1 4 7
4 6 8
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_19>
```

## Source Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXLINES 5000
#define MAXLEN 100

char *lineptr[MAXLINES];

int readlines(char *lineptr[], int maxlines);
void writelines(char *lineptr[], int nlines);
void fieldSort(void *lineptr[], int left, int right, int field, int (*comp)(const char *, const char *));
int numcmp(const char *s1, const char *s2);
int foldcmp(const char *s1, const char *s2);

/*****
 * Function Name: main
 * Description: Entry point of the program.
 * Parameters:
 *   - argc: Number of command-line arguments.
 *   - argv: Array of strings containing the command-line arguments.
 * Returns:
 *   - 0 on successful execution, 1 on error.
 *****/
int main(int argc, char *argv[])
{
    int nlines, i;
    int field = 0;
    int numeric = 0;
    int reverse = 0;
    int fold = 0;

    // Check command-line arguments for flags
    for (i = 1; i < argc; i++)
    {
        if (strcmp(argv[i], "-n") == 0)
            numeric = 1;
        else if (strcmp(argv[i], "-r") == 0)
            reverse = 1;
        else if (strcmp(argv[i], "-f") == 0)
            fold = 1;
        else if (strcmp(argv[i], "-k") == 0 && i + 1 < argc)
            field = atoi(argv[++i]);
    }

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0)
    {
        int i, j;

        fieldSort((void **)lineptr, 0, nlines - 1, field, (int (*)(const char *, const char
*)))(numeric ? numcmp : (fold ? foldcmp : strcmp));
        if (reverse)
        {
            // Reverse the sorted lines
            for (i = 0, j = nlines - 1; i < j; i++, j--)
            {
                void *temp = lineptr[i];
```

```

        lineptr[i] = lineptr[j];
        lineptr[j] = temp;
    }
}
writelines(lineptr, nlines);
return 0;
}
else
{
    printf("input too big to sort\n");
    return 1;
}
}

/*****
****
* Function Name: readlines
* Description: Reads lines from input and stores them in an array of pointers to strings.*
* Parameters:
* - lineptr: Array of pointers to strings where the lines will be stored.
* - maxlines: Maximum number of lines to read.
* Returns:
* - The number of lines read.
****
*****/
int readlines(char *lineptr[], int maxlines)
{
    int nlines = 0;
    char line[MAXLEN];

    while (fgets(line, sizeof(line), stdin) != NULL)
    {
        line[strcspn(line, "\n")] = '\0'; // Remove trailing newline character
        lineptr[nlines] = strdup(line); // Store a copy of the line
        nlines++;
    }

    return nlines;
}

/*****
****
* Function Name: writelines
* Description: Writes the lines stored in an array of pointers to strings to the output.*
* Parameters:
* - lineptr: Array of pointers to strings containing the lines to write.
* - nlines: Number of lines to write.
* Returns:
* - None.
****
*****/
void writelines(char *lineptr[], int nlines)
{
    int i;
    for (i = 0; i < nlines; i++)
    {

```

```

        printf("%s\n", lineptr[i]);
    }
}

/*****
****
* Function Name: fieldSort
* Description: Sorts an array of pointers based on a specific field in each line.
* Parameters:
* - lineptr: Array of pointers to be sorted.
* - left: Left index of the array to start sorting.
* - right: Right index of the array to end sorting.
* - field: Field number to consider for sorting (1-based indexing).
* - comp: Comparison function for determining the order of elements.
* Returns: None.
****
****/
void fieldSort(void *v[], int left, int right, int field, int (*comp)(const char *, const char *))
{
    int i, j, count;
    void *temp;
    char *line1, *line2;
    char *token1, *token2;

    for (i = left; i < right; i++)
    {
        for (j = i + 1; j <= right; j++)
        {
            line1 = strdup((char *)v[i]); // Create a copy of line1
            line2 = strdup((char *)v[j]); // Create a copy of line2

            // Extract fields from lines
            count = 1;
            token1 = strtok(line1, " "); // Tokenize line1 by space

            while (count < field && token1 != NULL)
            {
                token1 = strtok(NULL, " "); // Move to the next field
                count++;
            }

            count = 1;
            token2 = strtok(line2, " "); // Tokenize line2 by space

            while (count < field && token2 != NULL)
            {
                token2 = strtok(NULL, " "); // Move to the next field
                count++;
            }

            // Compare field values
            if ((*comp)(token1, token2) > 0)
            {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }
        }
    }
}

```

```

        free(line1); // Free the memory allocated for line1
        free(line2); // Free the memory allocated for line2
    }
}

/*****
 * Function Name: numcmp
 * Description: Compares two strings numerically.
 * Parameters:
 *   - s1: Pointer to the first string to compare.
 *   - s2: Pointer to the second string to compare.
 * Returns:
 *   - -1 if s1 < s2, 0 if s1 == s2, 1 if s1 > s2.
 *****/
int numcmp(const char *s1, const char *s2)
{
    double v1 = atof(s1);
    double v2 = atof(s2);

    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}

/*****
 * Function Name: foldcmp
 * Description: Compares two strings case-insensitively.
 * Parameters:
 *   - s1: First string to compare.
 *   - s2: Second string to compare.
 * Returns:
 *   - Negative value if s1 is less than s2.
 *   - Positive value if s1 is greater than s2.
 *   - 0 if s1 and s2 are equal.
 *****/
int foldcmp(const char *s1, const char *s2)
{
    for (; *s1 && *s2; s1++, s2++)
    {
        if (tolower(*s1) != tolower(*s2))
            return tolower(*s1) - tolower(*s2);
    }

    return *s1 - *s2;
}

```