Learning summary:
Study: 5.5 Hours
Exercises: 2 Hours

Documentation of Day_19

Exercise 5-14:

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

Source Code:

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

#define MAXLINES 5000 /* max #lines to be sorted */
#define MAXLEN 1000   /* max length of any input line */

char *lineptr[MAXLINES]; /* pointers to text lines */

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

/* sort input lines */
int main(int argc, char *argv[])
{
  int nlines; /* number of input lines read */
  int numeric = 0; /* 1 if numeric sort */
  int reverse = 0; /* 1 if reverse sort */

        int i;
  for(i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-n") == 0)
      numeric = 1;
    else if (strcmp(argv[i], "-r") == 0)
      reverse = 1;
  }

  if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
    my_qsort((void**)lineptr, 0, nlines-1, (int (*)(void*,void*))(numeric ? numcmp : strcmp), reverse);
    writelines(lineptr, nlines);
    return 0;
  } else {
    printf("input too big to sort\n");
    return 1;
  }
}

/* readlines: read input lines */
int readlines(char *lineptr[], int maxlines)
{
   int len, nlines;
   char *p, line[MAXLEN];

   nlines = 0;
   while ((len = getline(line, MAXLEN)) > 0) {
      if (nlines >= maxlines || (p = malloc(len)) == NULL)
         return -1;
      else {
         line[len-1] = '\0'; /* remove newline character */
         strcpy(p, line);
         lineptr[nlines++] = p;
      }
   }
   return nlines;
}

/* writelines: write output lines */
void writelines(char *lineptr[], int nlines)
{
          int i;
   for(i = 0; i < nlines; i++)
      printf("%s\n", lineptr[i]);
}

/* qsort: sort v[left]...v[right] into increasing order */
void my_qsort(void *v[], int left, int right, int (*comp)(void *, void *), int reverse)
{
   int i, last;
   void swap(void *v[], int i, int j);

   if (left >= right)
      return;

   swap(v, left, (left + right)/2);
   last = left;
   for (i = left + 1; i <= right; i++) {
      if ((reverse && (*comp)(v[i], v[left]) > 0) || (!reverse && (*comp)(v[i], v[left]) < 0))
         swap(v, ++last, i);
   }
   swap(v, left, last);
   my_qsort(v, left, last-1, comp, reverse);
   my_qsort(v, last+1, right, comp, reverse);
}
```

```
/* numcmp: compare s1 and s2 numerically */
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;
}

/* swap: interchange v[i] and v[j] */
void swap(void *v[], int i, int j)
{
    void *temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}

int getline(char *s, int lim)
{
    int c, i;

    for (i = 0; i < lim-1 && (c = getchar()) != EOF && c != '\n'; ++i)
        s[i] = c;

    if (c == '\n') {
        s[i] = c;
        ++i;
    }

    s[i] = '\0';
    return i;
}
```
Here are the key points of the modified program that I have changed and how it performs:

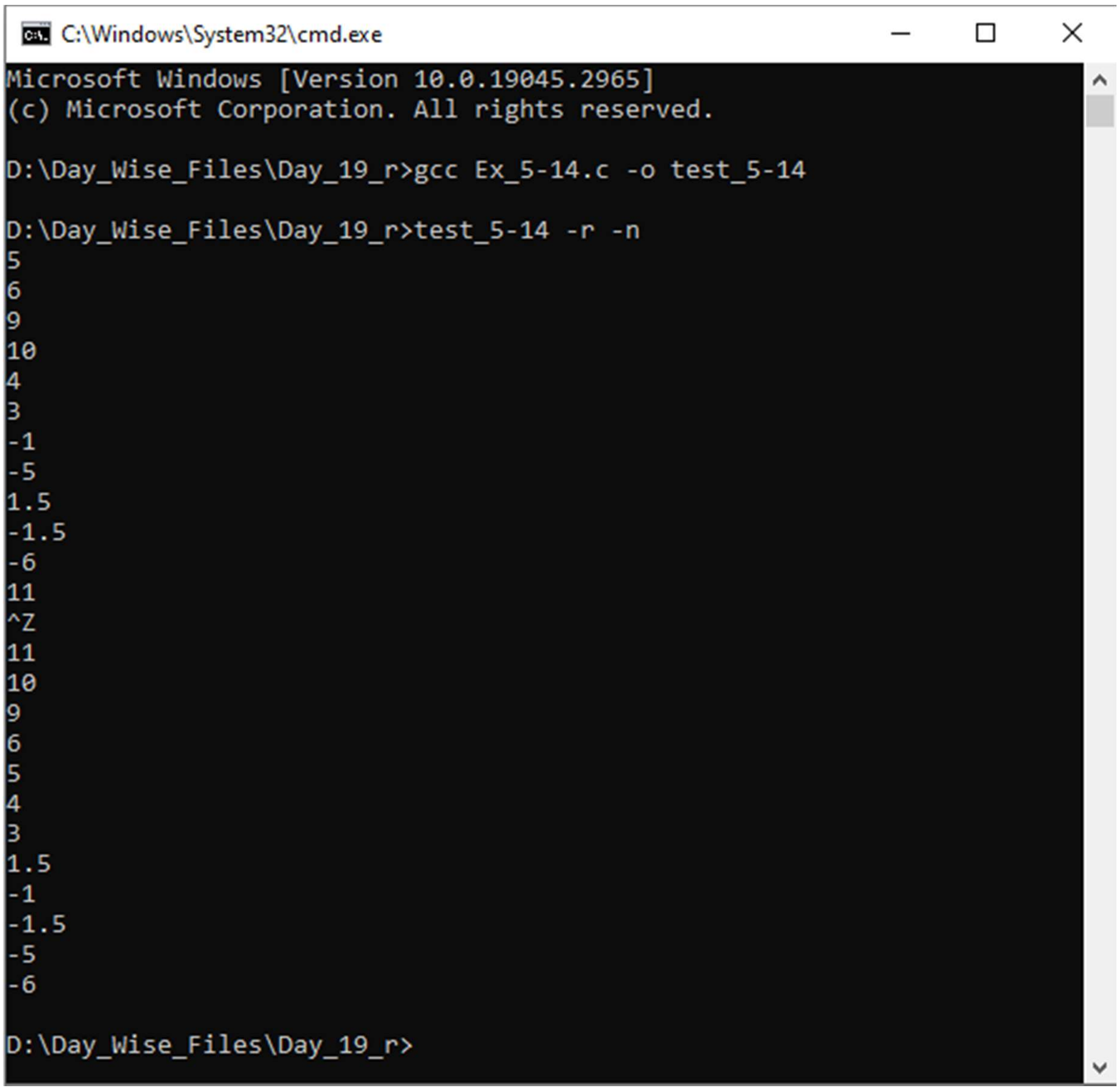
1. **Command-Line Arguments:** The program takes command-line arguments to determine the sorting behavior. It checks for the -n flag to perform numeric sorting and the -r flag to indicate reverse sorting. These flags are set as boolean variables (numeric and reverse) to control the sorting logic.
2. **Sorting Function (qsort):** The qsort function is responsible for sorting the array of strings (lineptr) using the QuickSort algorithm. It takes additional parameters int (*comp)(void *, void *) and int reverse to handle the comparison and reverse sorting. The comp function pointer allows for

flexible comparison based on whether numeric or lexicographic sorting is needed. The reverse flag determines the direction of sorting.

3. **Comparison Functions:** The program includes a comparison function numcmp to compare strings numerically (-n flag) and the default strcmp for lexicographic comparison. The numcmp function uses atof to convert the strings to numbers and perform the comparison.
4. **Swapping Pointers:** The swap function is responsible for swapping two pointers in the array. It is used within the qsort function to rearrange the elements during the sorting process.
5. **Reading and Writing Lines:** The program includes functions readlines and writelines to read input lines from the user and print the sorted lines, respectively. readlines reads lines from input and stores them in the lineptr array. writelines prints the sorted lines stored in the lineptr array.

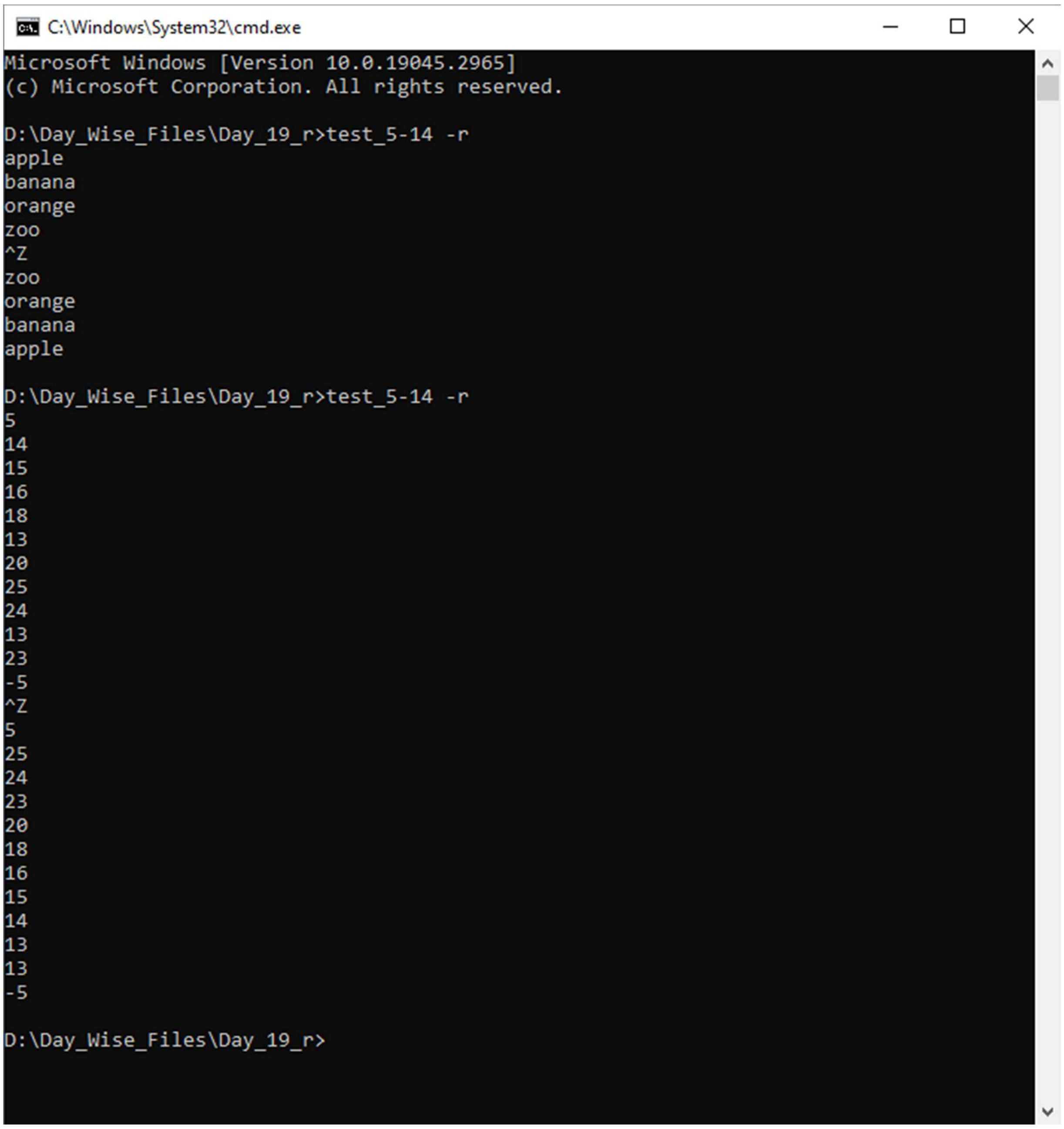
Inputs and Outputs:

1.



```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

D:\Day_Wise_Files\Day_19_r>gcc Ex_5-14.c -o test_5-14

D:\Day_Wise_Files\Day_19_r>test_5-14 -r -n
5
6
9
10
4
3
-1
-5
1.5
-1.5
-6
11
^Z
11
10
9
6
5
4
3
1.5
-1
-1.5
-5
-6

D:\Day_Wise_Files\Day_19_r>
```

2.



```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

D:\Day_Wise_Files\Day_19_r>test_5-14 -r
apple
banana
orange
zoo
^Z
zoo
orange
banana
apple

D:\Day_Wise_Files\Day_19_r>test_5-14 -r
5
14
15
16
18
13
20
25
24
13
23
-5
^Z
5
25
24
23
20
18
16
15
14
13
13
-5

D:\Day_Wise_Files\Day_19_r>
```

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.

To add the option -f in the program to fold upper and lower case together, I have made the following modifications to previous program:

Update the main function:

- Add a boolean variable fold and initialize it to 0.
- Inside the for loop that processes the command-line arguments, check if the current argument is -f and set the fold variable to 1.

Modify the my_qsort function:

- Before performing the comparison inside the if statement, add a new condition to check if the fold variable is set.

If fold is set, convert both strings v[i] and v[left] to lowercase before comparing them.

Source Code:

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

#define MAXLINES 5000 /* max #lines to be sorted */
#define MAXLEN 1000   /* max length of any input line */

char *lineptr[MAXLINES]; /* pointers to text lines */

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

/* sort input lines */
int main(int argc, char *argv[]) {
   int nlines; /* number of input lines read */
   int numeric = 0; /* 1 if numeric sort */
   int reverse = 0; /* 1 if reverse sort */
   int fold = 0;    /* 1 if case folding */

   int i;
   for (i = 1; i < argc; i++) {
      if (strcmp(argv[i], "-n") == 0)
```

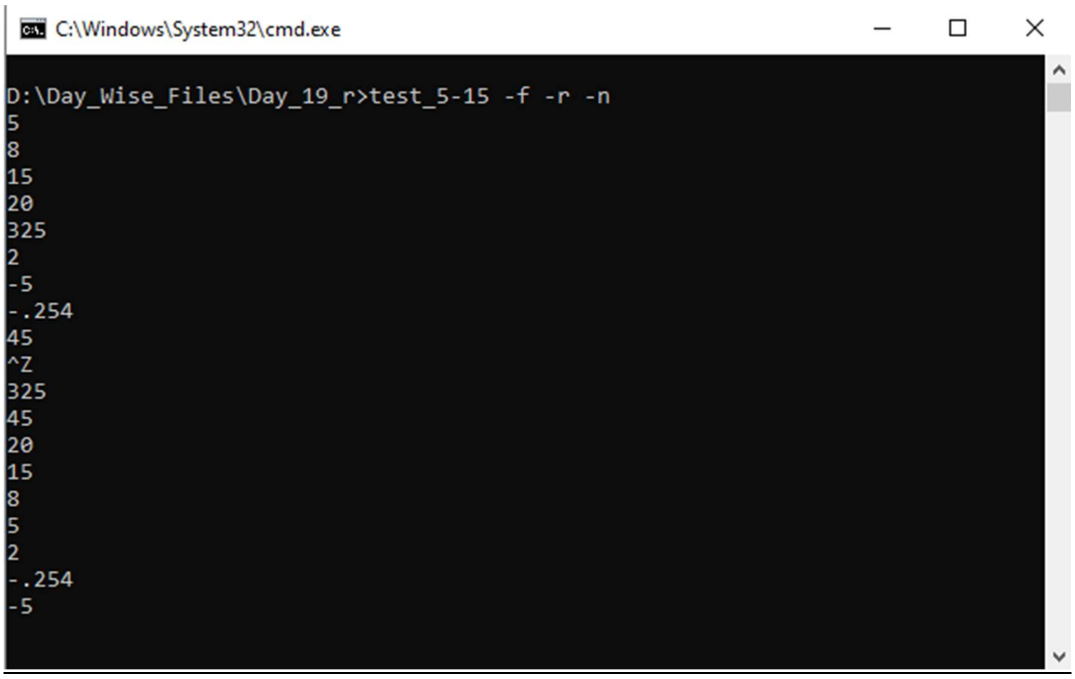
```
            numeric = 1;
        else if (strcmp(argv[i], "-r") == 0)
            reverse = 1;
        else if (strcmp(argv[i], "-f") == 0)
            fold = 1;
    }

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        my_qsort((void **)lineptr, 0, nlines - 1, (int (*)(void *, void *, int))(numeric ? numcmp : strcasecmp),
reverse, fold);
        writelines(lineptr, nlines);
        return 0;
    }
    else {
        printf("input too big to sort\n");
        return 1;
    }
}

/* readlines: read input lines */
int readlines(char *lineptr[], int maxlines) {
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0) {
        if (nlines >= maxlines || (p = malloc(len)) == NULL)
            return -1;
        else {
            line[len - 1] = '\0'; /* remove newline character */
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    }
    return nlines;
}

/* writelines: write output lines */
void writelines(char *lineptr[], int nlines) {
    int i;
    for (i = 0; i < nlines; i++)
        printf("%s\n", lineptr[i]);
}
```

```
/* qsort: sort v[left]...v[right] into increasing order */
void my_qsort(void *v[], int left, int right, int (*comp)(void *, void *, int), int reverse, int fold) {
   int i, last;
   void swap(void *v[], int i, int j);

   if (left >= right)
      return;

   swap(v, left, (left + right) / 2);
   last = left;
   for (i = left + 1; i <= right; i++) {
      if ((reverse && (*comp)(v[i], v[left], fold) > 0) || (!reverse && (*comp)(v[i], v[left], fold) < 0))
         swap(v, ++last, i);
   }
   swap(v, left, last);
   my_qsort(v, left, last - 1, comp, reverse, fold);
   my_qsort(v, last + 1, right, comp, reverse, fold);
}

/* numcmp: compare s1 and s2 numerically */
int numcmp(const char *s1, const char *s2, int fold) {
   double v1 = atof(s1);
   double v2 = atof(s2);

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

/* swap: interchange v[i] and v[j] */
void swap(void *v[], int i, int j) {
   void *temp = v[i];
   v[i] = v[j];
   v[j] = temp;
}

int getline(char *s, int lim) {
   int c, i;

   for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
      s[i] = c;
```

```
    if (c == '\n') {
        s[i] = c;
        ++i;
    }

    s[i] = '\0';
    return i;
}
```

Input and Output:

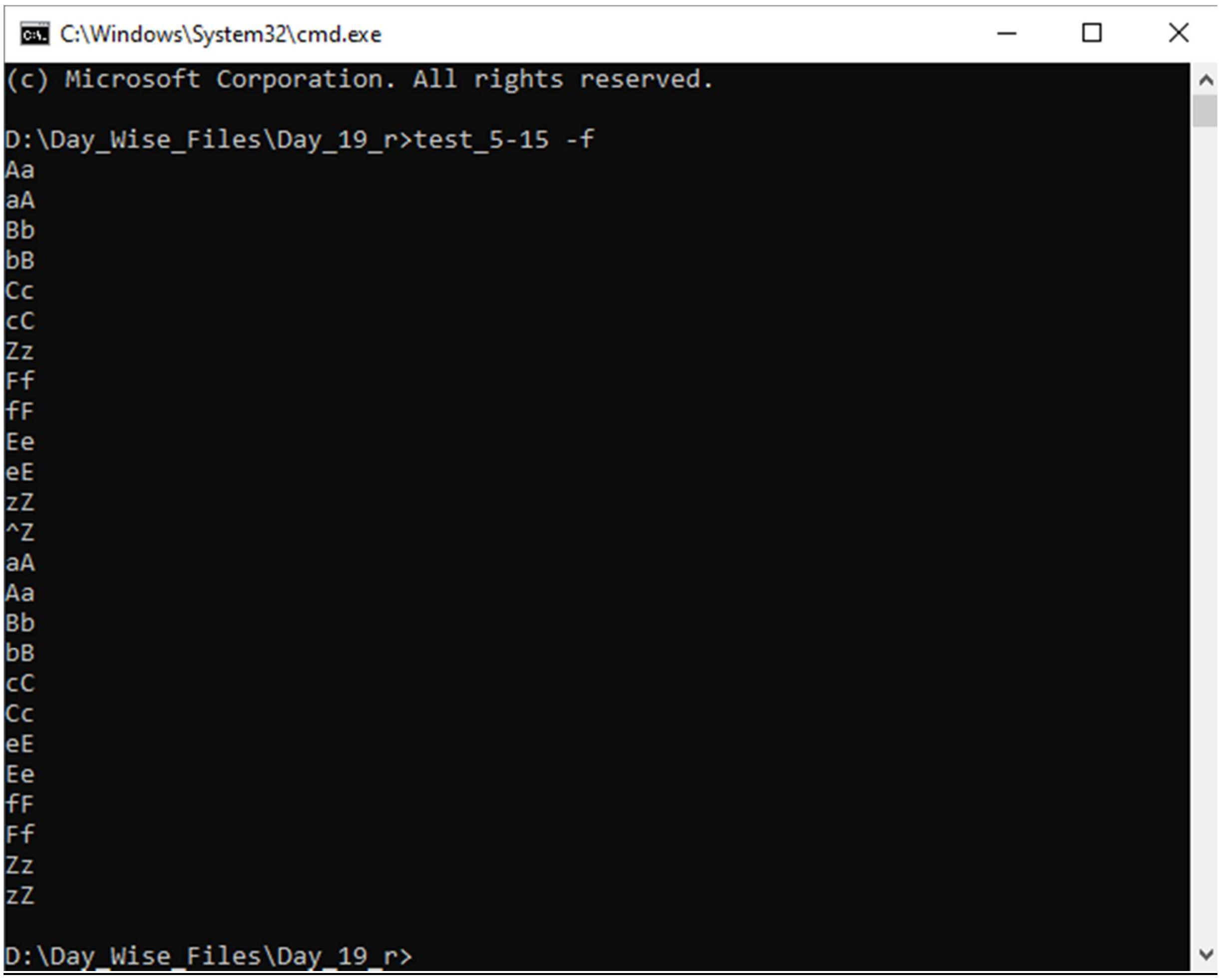
1.



C:\Windows\System32\cmd.exe

```
D:\Day_Wise_Files\Day_19_r>test_5-15 -f -r -n
5
8
15
20
325
2
-5
-.254
45
^Z
325
45
20
15
8
5
2
-.254
-5
```

2.



```
(c) Microsoft Corporation. All rights reserved.

D:\Day_Wise_Files\Day_19_r>test_5-15 -f
Aa
aA
Bb
bB
Cc
cC
Zz
Ff
fF
Ee
eE
zZ
^Z
aA
Aa
Bb
bB
cC
Cc
eE
Ee
fF
Ff
Zz
zZ

D:\Day_Wise_Files\Day_19_r>
```