Learning summary:
Study: 6 Hours
Exercises: 2 Hours

Day 16

Exercise 5-7:

Question:

Rewrite readlines to store lines in an array supplied by main, rather than calling alloc to maintain storage. How much faster is the program?

Solution:

Source Code:

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

#define MAXLINES 5000
#define MAXLEN 1000 /* max length of any input line */
char *lineptr[MAXLINES];

int readlines(char *lineptr[], char *lines, int maxlines);
void writelines(char *lineptr[], int nlines);
void qsort(char *lineptr[], int left, int right);
int getline(char s[], int lim);

#define ARRAYSIZE 10000

main()
{
   int nlines; /* number of input lines read */
   char holder[MAXLEN];
   if ((nlines = readlines(lineptr, holder, MAXLINES)) >= 0)
   {
      qsort(lineptr, 0, nlines - 1);
      writelines(lineptr, nlines);
      return 0;
   }
   else
   {
      printf("error: input too big to sort\n");
      return 1;
   }
}

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;
}

/* readlines: read input lines */
int readlines(char *lineptr[], char *lines, int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];
    nlines = 0;
    p = lines; // Set the initial pointer to the start of the supplied holder array
    while ((len = getline(line, MAXLEN)) > 0)
    {
        if (nlines >= MAXLINES || (p - lines + len) > ARRAYSIZE)
            return -1;
        else
        {
            line[len - 1] = '\0'; /* delete newline(\n) */
            strcpy(p, line);
            lineptr[nlines++] = p;
            p += len; // Move the pointer to the next available position
        }
    }

    return nlines;
}

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

/* swap: interchange v[i] and v[j] */
void swap(char *v[], int i, int j)
```

```
{
   char *temp;
   temp = v[i];
   v[i] = v[j];
   v[j] = temp;
}

/* qsort: sort v[left]...v[right] into increasing order */
void qsort(char *v[], int left, int right)
{
   int i, last;
   void swap(char *v[], int i, int j);
   if (left >= right) /* do nothing if array contains */
      return;       /* fewer than two elements */
   swap(v, left, (left + right) / 2);
   last = left;
   for (i = left + 1; i <= right; i++)
      if (strcmp(v[i], v[left]) < 0)
         swap(v, ++last, i);
   swap(v, left, last);
   qsort(v, left, last - 1);
   qsort(v, last + 1, right);
}
```

Functions:

Readlines() function :

To rewrite the readlines function to store lines in an array supplied by main instead of using alloc, I have modified the function signature and the way lines are stored. Here's the modified version of the readlines function:

In the modified readlines function, we compare the current position (p) with the start of the lines array and the available space (ALLOCSIZE) to ensure we don't exceed the array bounds.

The performance improvement of this modified version depends on various factors, including the input size, the number of lines, and the available memory. Since we're no longer calling alloc for each line, the modified version may have better performance in terms of memory management. However, the overall speed improvement may not be significant unless the original program was limited by memory allocation. The impact on performance can vary based on the specific use case and system configuration.

main() function :

I have make changes in the main function to use the modified readlines function that stores lines in the array supplied by main. In the modified main function, we declare an array lines with dimensions [MAXLEN] to store the lines. . This array is passed as an argument to the readlines function. By passing lines as the second argument to readlines, we supply the array to store the lines. The modified readlines function will store the lines directly in this array. The modifications are primarily related to the changes in the readlines function and passing the lines array to it.

The modified code stores lines directly in the lines array supplied by main, the alloc function is no longer necessary. The purpose of the alloc function in the original code was to allocate memory for each line using dynamic memory allocation. However, since we are now directly storing the lines in the lines array, there is no need for dynamic memory allocation.

Input & Output:

```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_16\Ex_5-7.exe
abdcshi
bdcohsf
aahfhfsf
aaahfsf
123464
11343
111
1212454
1111564141
^Z
111
1111564141
11343
1212454
123464
aaahfsf
aahfhfsf
abdcshi
bdcohsf

--------------------------------
Process exited after 29.78 seconds with return value 0
Press any key to continue . . .
```

Exercise 5-9:

Question:

Rewrite the routines day_of_year and month_day with pointers instead of indexing

Solution:

Source Code:

```
#include <stdio.h>

static char daytab[2][13] = {
   {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
   {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

int day_of_year(int year, int month, int day);
void month_day(int year, int yearday, int *pmonth, int *pday);
char *month_name(int n);

int main()
```

```
{
   int year = 2023;
   int month = 6;
   int day = 15;

   int yearday = day_of_year(year, month, day);
   printf("Day of the year: %d\n", yearday);

   int retrieved_month, retrieved_day;
   month_day(year, yearday, &retrieved_month, &retrieved_day);
   printf("Month: %s, Day: %d\n", month_name(retrieved_month), retrieved_day);

   return 0;
}

int day_of_year(int year, int month, int day)
{
   int leap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
   char *p = *(daytab + leap);

   while (--month)
      day += *++p;

   return day;
}

void month_day(int year, int yearday, int *pmonth, int *pday)
{
   int leap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
   char *p = *(daytab + leap);

   while (yearday > *++p)
      yearday -= *p;

   *pmonth = p - *(daytab + leap);
   *pday = yearday;
}

char *month_name(int n)
{
   static char *name[] = {
      "Illegal month",
      "January", "February", "March",
      "April", "May", "June",
```

```
    "July", "August", "September",
    "October", "November", "December"
  };
  return (n < 1 || n > 12) ? name[0] : name[n];
}
```

Functions:

main() function :

Demonstrates the usage of the above functions by calculating and printing the day of the year, month, and day for a given date.

day_of_year function :

Calculates the day of the year given a specific date (year, month, day).

month_day() function :

Converts a day of the year to the corresponding month and day.

***month_name() Function:**

Returns the name of a month based on its number.

main():

The main() function invokes the **process_input()** and **draw_histogram()** functions and the histogram is printed of the frequencies of different characters in its input.

Input & Output: