

Day 16 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

****(Reading time 6 hour and problem solving, debug time and doc 2 hour 50 min)****

.....

1. Exercise 5-7:

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

Analysis:

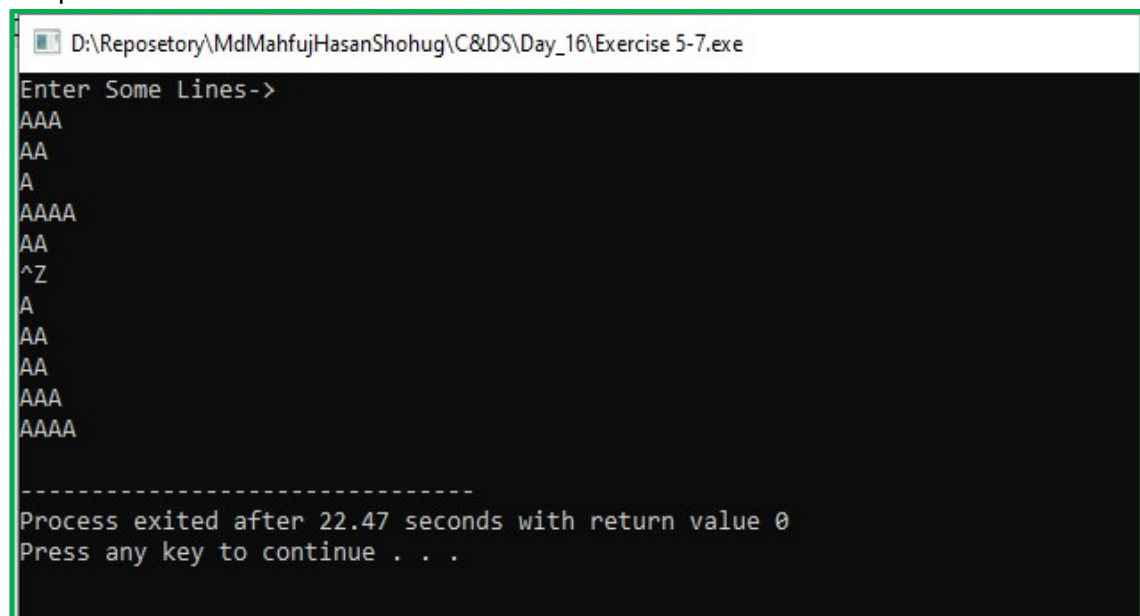
For the specific part change according to the question I just remove the alloc function and for this specific part of change I just make change readlines() function and main() function. And other I give as it is on the book function and also code those are rewrite the update version.

For the second question:

I added a int nextFreeLoc pointer parameter, which represents the pointer provided by next free location, to the updated readline function. Instead of allocating memory with alloc, we just copy each line directly to the appropriate array line. I configured lineptr to point at the line that lines up with line.

My program runs more quickly by using alloc and storing directly in the provided array since the overhead of memory allocation and deallocation is removed.

Outputs:



```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-7.exe
Enter Some Lines->
AAA
AA
A
AAAA
AA
^Z
A
AA
AA
AAA
AAAA

-----
Process exited after 22.47 seconds with return value 0
Press any key to continue . . .
```

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-7.exe
Enter Some Lines->
11
12
222
333
3333
^Z
11
12
222
333
3333
-----
Process exited after 21.23 seconds with return value 0
Press any key to continue . . .

D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-7.exe
Enter Some Lines->
AAAABBB
AAAABBB
AAAABBB
^Z
AAAABBB
AAAABBB
AAAABBB
-----
Process exited after 23.63 seconds with return value 0
Press any key to continue . . .

D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-7.exe
Enter Some Lines->
hal fasjdfha
fasflda ;sdf
gfdsjljgsfdvn;nva[sd
fdsavbjdn
adflknsfjna ;psdvncavbnirf
flgbjasdf;gnafg
gvf;dslvjnfg
^Z
adflknsfjna ;psdvncavbnirf
fasflda ;sdf
fdsavbjdn
flgbjasdf;gnafg
gfdsjljgsfdvn;nva[sd
gvf;dslvjnfg
hal fasjdfha
-----
Process exited after 14.65 seconds with return value 0
Press any key to continue . . .
```

Source Code:

```

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

#define MAXLINES 10000
char *lineptr[MAXLINES];
int readlines(char *lineptr[], int nlines, char *storStr, int *nextFreeLoc);
void writelines(char *lineptr[], int nlines);

#define MAXLEN 10000

void qsort(char *lineptr[], int left, int right);
void swap(char *v[], int i, int j);

#define MAXSTRLEN 10000

/*****
** Function: main
** Description: Entry point of the program.
** Inputs:
**   - None
** Outputs:
**   - Returns 0 on success, non-zero on failure.
*****/
int main(int argc, char *argv[])
{
    int nlines;
    char storStr[MAXSTRLEN];
    int nextFreeLoc = 0;
    printf("Enter Some Lines->\n");
    if ((nlines = readlines(lineptr, MAXLINES, storStr, &nextFreeLoc)) >= 0)
    {
        qsort(lineptr, 0, nlines - 1);
        writelines(lineptr, nlines);
        return 0;
    }
    else
    {
        printf("error: input too big to sort\n");
        return 1;
    }
}

/*****
** Function: getline
** Description: Reads a line from input and stores it in the provided string.
** Inputs:
**   - s: Pointer to the string where the line will be stored.
**   - lim: Maximum length of the string.
** Outputs:
**   - Returns the length of the line read (excluding the null terminator).
*****/
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;
}

```

```

/*****
** Function: readlines
** Description: Reads lines from input and stores them in an array of strings.
** Inputs:
**   - lineptr: Array of string pointers to store the lines.
**   - maxlines: Maximum number of lines that can be stored.
**   - storStr: Buffer to store the lines.
**   - nextFreeLoc: Pointer to the next free location in the buffer.
** Outputs:
**   - Returns the number of lines read on success, -1 on failure.
*****/
int readlines(char *lineptr[], int maxlines, char *storStr, int *nextFreeLoc) // Change here for parameter pointer
{
    int len, nlines;
    char line[MAXLEN];
    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
    {
        if (nlines >= maxlines || (MAXSTRLEN - *nextFreeLoc) < len) //add condition for replacing alloc function
            return -1;
        else
        {
            line[len - 1] = '\0';
            strcpy(&storStr[*nextFreeLoc], line);
            lineptr[nlines] = &storStr[*nextFreeLoc];
            *nextFreeLoc += len;
            nlines++;
        }
    }
    return nlines;
}

/*****
** Function: writelines
** Description: Writes the lines stored in an array of strings to the output.
** Inputs:
**   - lineptr: Array of string pointers containing the lines.
**   - nlines: Number of lines to write.
** Outputs:
**   - None
*****/
void writelines(char *lineptr[], int nlines)
{
    while (nlines--)
    {
        printf("%s\n", *lineptr++);
    }
}

/*****
** Function: qsort
** Description: Sorts an array of strings using the quicksort algorithm.
** Inputs:
**   - v: Array of string pointers to be sorted.
**   - left: Starting index of the array.
**   - right: Ending index of the array.
** Outputs:
**   - None
*****/
void qsort(char *v[], int left, int right)
{
    int i, last;
    if (left >= right)
        return;
    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);
    }

/*****
** Function: swap
** Description: Swaps two string pointers.
** Inputs:
**   - v: Array of string pointers.
**   - i: Index of the first pointer.
**   - j: Index of the second pointer.
** Outputs:
**   - None
*****/
void swap(char *v[], int i, int j)
{
    char *temp;
    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}

```

2. Exercise 5-9:

Problem: Rewrite the routines `day_of_year` and `month_day` with pointers instead of Indexing.

Analysis: Here in this code I just change the `day_of_year` and `month_day` function and using the pointer in a condition and loop with same output I given the code, other function and code I give from the book as it is.

Here I am set pointer as a day and month and 165th day the date.

- The day of the year, or -1 if the date is invalid.
- `*(*(daytab + leap) + i)` using this expression instated of indexing there for used to access a specific element in a two dimensional array or an array of pointer with given the indicate leap and i.

Outputs:

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-9.exe
Number of day in this year = 165

The date is: 6th month, 14th day

-----
Process exited after 0.03315 seconds with return value 0
Press any key to continue . . .
```

Also same day for leap year 2024 output:

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-9.exe
Number of day in this year = 166

The date is: 6th month, 13th day

-----
Process exited after 0.03068 seconds with return value 0
Press any key to continue . . .
```

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_16\Exercise 5-9.exe
Number of day in this year = -1

The date is: -1th month, -1th day

-----
Process exited after 0.03048 seconds with return value 0
Press any key to continue . . .
```

For invalid month 13 and day 32 and year day 465 and this is return -1.

Source code:

```
#include <stdio.h>

static char daytab[][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);

//Main function
```

```

int main(int argc, char *argv[])
{
    int a, b;

    printf("Number of day in this year = %d\n", day_of_year(2023, 6, 14)); // The date is 14th june 2023 as an input

    month_day(2023, 165, &a, &b); //pointer as a day and month and 165th day the date.
    printf("The date is: %dth month, %dth day\n", a, b);

    return 0;
}
/**
 * Calculates the day of the year based on the given date.
 *
 * @param year The year.
 * @param month The month.
 * @param day The day.
 * @return The day of the year, or -1 if the date is invalid.
 */
int day_of_year(int year, int month, int day)
{
    int i, leap;
    leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
    if (year >= 1582 && month > 0 && month <= 12 && day > 0 && day <= (*(daytab + leap) + month)) // Using Pointer
    instated of indexing
    {
        for (i = 1; i < month; i++)
        {
            day += (*(daytab + leap) + i); // Using Pointer instated of indexing
        }

        return day;
    }
    else
        return -1;
}

/**
 * Calculates the month and day based on the given day of the year.
 *
 * @param year The year.
 * @param yearday The day of the year.
 * @param pmonth Pointer to the variable to store the month.
 * @param pday Pointer to the variable to store the day.
 */
void month_day(int year, int yearday, int *pmonth, int *pday)
{
    int i, leap;
    leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
    if (year >= 1582 && yearday > 0 && (yearday <= 366 && leap || yearday <= 365))
    {
        for (i = 1; yearday > (*(daytab + leap) + i); i++) // Using Pointer instated of indexing
        {
            yearday -= (*(daytab + leap) + i); // Using Pointer instated of indexing
        }
        *pmonth = i;
        *pday = yearday;
    }
    else
        *pmonth = *pday = -1;
}

```