**Header.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FILE\_NAME : header.h

\* DESCRIPTION : the file contains the header files ,

\* macro definitions and the function prototypes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HEADER\_FILES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MACRO DEFINITIONS

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MAX 20

#define SIZE 100 // Increased buffer size for reading full lines

#define SUCCESS 1

#define FAILURE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION\_PROTOTYPES

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_string(char \*array[], int i);

int binary\_search(char \*array[], char \*find, int n);

int bubble\_sort(char \*array[], int n);

int delete\_line(char \*array[], int \*n, char \*line);

void write\_to\_file(char \*array[], int n);

**makefile**

SRC = ./../src/

OBJ = ./../obj/

BIN = ./../bin/

GFLAGS = gcc

INC = ./../include/

CFLAGS = -c -g -Wall

OFLAGS = -lm -o

IFLAGS = -I./../include/

VFLAGS = v --tool=memcheck --leak-check=yes --show-reachable=yes

EXECS = $(BIN)final

$(BIN)final :$(OBJ)main.o $(OBJ)function.o

$(GFLAGS) $(OFLAGS)final $(OBJ)main.o $(OBJ)function.o

mv final $(BIN)

$(OBJ)main.o:$(SRC)main.c $(INC)header.h

$(GFLAGS) $(CFLAGS) $(SRC)main.c $(IFLAGS)

mv main.o $(OBJ)

$(OBJ)function.o:$(SRC)function.c $(INC)header.h

$(GFLAGS) $(CFLAGS) $(SRC)function.c $(IFLAGS)

mv function.o $(OBJ)

clean:

rm $(OBJ)\*

rm $(BIN)final

valgrind:$(EXECS)

valgrind $(VFLAGS) $(BIN)final

**src - function.c**

#include "header.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: get\_string

\* DESCRIPTION: gets a line from standard input to the i-th

\* element of the array

\* ARGUMENTS: Array of strings to be stored

\* RETURNS: returns SUCCESS or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_string(char \*array[], int i)

{

int len;

char buff[SIZE];

memset(buff, 0, SIZE); // Initializing the buffer to zero

fgets(buff, SIZE - 1, stdin); // Get input from user

len = strlen(buff);

if ('\n' == buff[len - 1]) // Replacing '\n' with '\0'

{

buff[len - 1] = '\0';

}

else

{

while (getchar() != '\n') // Cleaning the buffer

{

;

}

}

// Allocating memory for the i-th element

array[i] = (char \*)malloc(len \* sizeof(char));

if (NULL == array[i])

{

printf("Memory Allocation Failed\n");

return FAILURE;

}

strcpy(array[i], buff); // Copying the buffer to array[i]

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: bubble\_sort

\* DESCRIPTION: Sorts the list of strings in lexicographical order

\* ARGUMENTS: Array of strings and number of strings

\* RETURNS: SUCCESS after sorting

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int bubble\_sort(char \*array[], int n)

{

int i, j;

int SWAP;

char \*temp;

for (i = 0; i < (n - 1); i++)

{

SWAP = 0;

for (j = 0; j < (n - 1 - i); j++)

{

if (strcmp(array[j], array[j + 1]) > 0)

{

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

SWAP = 1;

}

}

if (SWAP == 0)

{

break; // No swap, array is already sorted

}

}

printf("\n\nBUBBLE SORT DONE\n");

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: delete\_line

\* DESCRIPTION: Searches for a specific line and deletes it

\* ARGUMENTS: Array of strings, number of strings, and the line to delete

\* RETURNS: SUCCESS if deleted, FAILURE if not found

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int delete\_line(char \*array[], int \*n, char \*line)

{

int i;

for (i = 0; i < \*n; i++)

{

if (strcmp(array[i], line) == 0) // Match found

{

free(array[i]);

for (int j = i; j < \*n - 1; j++) // Shift elements

{

array[j] = array[j + 1];

}

(\*n)--; // Reduce the number of elements

return SUCCESS;

}

}

return FAILURE; // Line not found

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: binary\_search

\* DESCRIPTION: Searches for a line in the given list

\* ARGUMENTS: Array of strings, search string, and number of elements

\* RETURNS: Index of element or FAILURE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int binary\_search(char \*array[], char \*find, int n)

{

int BEG = 0;

int END = n - 1;

int MID;

while (BEG <= END)

{

MID = (BEG + END) / 2;

if (strcmp(find, array[MID]) == 0)

{

return MID;

}

if (strcmp(find, array[MID]) < 0)

{

END = MID - 1;

}

else

{

BEG = MID + 1;

}

}

return FAILURE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION NAME: write\_to\_file

\* DESCRIPTION: Writes the final list of strings to a file

\* ARGUMENTS: Array of strings and number of elements

\* RETURNS: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void write\_to\_file(char \*array[], int n)

{

FILE \*file = fopen("../data/out.txt", "w");

if (file == NULL)

{

printf("Error opening file for writing\n");

return;

}

for (int i = 0; i < n; i++)

{

fprintf(file, "%s\n", array[i]);

}

fclose(file);

printf("\n\nFinal list written to out.txt\n");

}

**Src – main.c**

#include "header.h"

int main()

{

char \*array[MAX]; // Initialization of the array

char search\_string[MAX];

int n, i, ret;

printf("\nHow many lines do you want to enter: ");

scanf("%d", &n); // Get input from user

while ('\n' != getchar()) // Cleaning the buffer

{

;

}

printf("\n\nEnter lines:\n");

for (i = 0; i < n; i++)

{

get\_string(array, i); // Get each line

}

printf("\n\nThe entered lines are:\n");

for (i = 0; i < n; i++)

{

printf("%s\n", array[i]); // Displaying the input lines

}

bubble\_sort(array, n); // Bubble sort the array of strings

printf("\n\nThe sorted lines are:\n");

for (i = 0; i < n; i++)

{

printf("%s\n", array[i]); // Displaying the sorted lines

}

// Ask user for line to delete

printf("\n\nEnter line to be searched and deleted: ");

fgets(search\_string, MAX, stdin);

int len = strlen(search\_string);

if ('\n' == search\_string[len - 1])

{

search\_string[len - 1] = '\0'; // Replacing '\n' with '\0'

}

else

{

while (getchar() != '\n') // Clear the buffer

{

;

}

}

ret = delete\_line(array, &n, search\_string); // Deleting the line

if (ret == FAILURE)

{

printf("\n\nLine not found\n");

}

else

{

printf("\n\nLine deleted successfully\n");

}

// Write the final list of lines to the file

write\_to\_file(array, n);

// Free allocated memory

for (i = 0; i < n; i++)

{

free(array[i]);

}

return SUCCESS;

}

**Output:**

I am working in capgemini