COMPUTER PROGRAMMING

Assignment 2

Github repository link

https://github.com/ayushp2002/C-Programs/tree/main/Assignment2

Ayush Kumar

Section B | 202017B3622

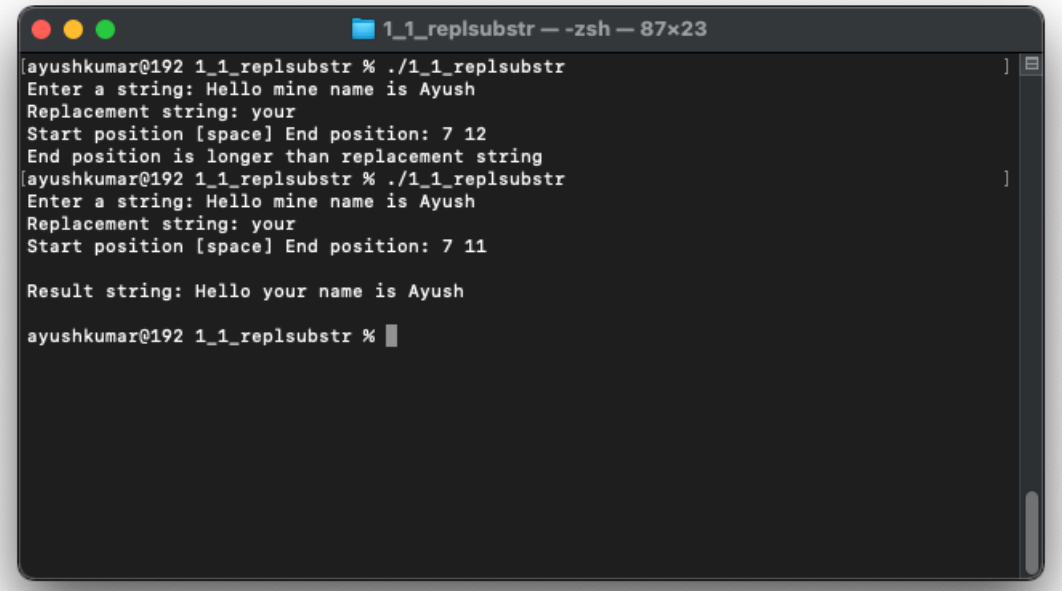
Table of Contents

Section 1 (Strings)

1. A string consists of a sentence. Write a program to replace a part of the string with another part of the string

Example:
Input: Give String: Mary goes to Delhi. She is doing the M.Tech at our college
Replaced String: has come from (issue with the output)
Position to replace: 5 to 11 (Start and end positions to replace)
Output: Mary comes from Delhi. She is doing the M.Tech at our college

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

char* repl_sub_str(char*, char*, int, int);

int main() {
    char inputstr[80], replstr[80];
    int startpos, endpos;

    printf("Enter a string: ");fgets(inputstr, 80, stdin);
    printf("Replacement string: ");fgets(replstr, 80, stdin);
    printf("Start position [space] End position: ");scanf("%d%d", &startpos, &endpos);
    // if the user gives an end position more than the replacement string can cover,
    // then give error and terminate program with RC 1
    if ((endpos-startpos) > strlen(replstr) - 1) {
        printf("End position is longer than replacement string\n");
        return 1;
    }

    printf("\nResult string: %s", repl_sub_str(inputstr, replstr, startpos, endpos));

    printf("\n");
    return 0;
}

/*
Replace the substring in the original string with the replacement string.
*/
char* repl_sub_str(char *str, char *repl, int start, int end) {
    char newstr[80];
    end--;
    // copy the text from original string
    for (int i = 0; i < start; i++) {
        newstr[i] = str[i];
    }
    // copy the text from replacement string
    for (int i = 0; i < end; i++) {
        newstr[start+i-1] = repl[i];
    }
    // copy the remaining text from original string
    for (int i = end; i < strlen(str); i++) {
        newstr[i] = str[i];
    }
    return newstr;
}
```



2. A paragraph consists of multiple lines. Write a program to count the number of articles in it.

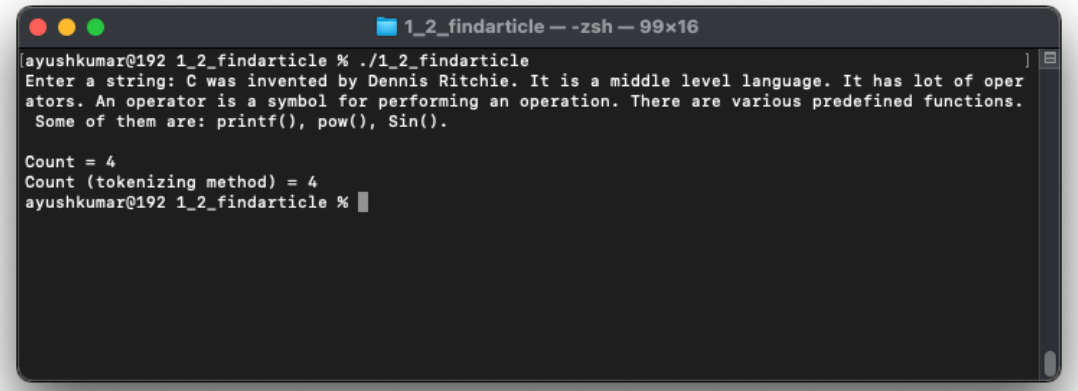
Input:
C was invented by Dennis Ritchie. It is **a** middle level language. It has lot of operators. **An** operator is **a** symbol for performing **an** operation. There are various predefined functions. Some of them are: printf(), pow(), Sin().
Output: 4

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

#define STRING_SIZE 300
#define WORD_SIZE 200

/*
    Tokenizing the string into words and then comparing is not the most efficient
    solution when we already know the substring. It is better to use simple comparisons.
    However, I have included both the methods for the below solution.
*/

int count_article(char*);
int count_article_by_splitting(char*);  // tokenizer method
int str_tolower(char*);

int main(int argc, char* argv[])
{
    char *str = malloc(STRING_SIZE * sizeof(char));
    printf("Enter a string: ");
    fgets(str, 300, stdin);

    str_tolower(str);   // strings are always passed by reference so no need to capture
returning value

    printf("\nCount = %d", count_article(str));
    printf("\nCount (tokenizing method) = %d", count_article_by_splitting(str));
```

```c
    printf("\n");

    return 0;
}

int count_article(char *str) {
    int count = 0, i = 0;
    int len = strlen(str) - 1;    // strlen counts the number of characters including escape
sequence.

    /* this kind of comparison with nested ifs is more optimized than using and(&),
       because if the first condition fails, it will not continue to the second one.
       this is better than computing complete logical expressions */

    // perform checks for first letter
    if (str[0] == 'a') {
        if (str[1] == 'n') {
            if (str[2] == ' ') {count++;i+=2;}    // "an"
        } else if (str[1] == ' ') {count++;i+=1;} // "a"
    } else if (str[0] == 't') {
        if (str[1] == 'h')
            if (str[2] == 'e')
                if (str[3] == ' ') {count++;i+=3;}   // "the"
    }

    // perform checks for intermediate letters
    /*
    If the current character is a space,
    then check the next character. If the next character is 'a', then check the next
    two characters. If the next two characters are 'an', then increment the count.
    If the next two characters are 'a', then increment the count. If the next
    character is 't', then check the next two characters. If the next two characters
    are 'the', then increment the count.
    */
    for (i = 0; i <= len; i++) {
        if (str[i] == ' ') {
            if (str[i+1] == 'a') {
                if (str[i+2] == 'n') {
                    if (str[i+3] == ' ') {count++;i+=2;}    // "an"
                } else if (str[i+2] == ' ') {count++;i+=1;} // "a"
            } else if (str[i+1] == 't') {
                if (str[i+2] == 'h')
                    if (str[i+3] == 'e')
                        if (str[i+4] == ' ') {count++;i+=3;}   // "the"
            }
        }
    }

    // perform checks for last letter
    if (str[len - 0] == 'n') {
        if (str[len - 1] == 'a')
            if (str[len - 2] == ' ')    // "an"
                count++;
    } else if (str[len - 0] == 'a') {
        if (str[len - 1] == ' ')    // "a"
            count++;
    } else if (str[len - 0] == 'e') {
```

```
        if (str[len - 1] == 'h')
            if (str[len - 2] == 't')
                if (str[len - 3] == ' ')    // "the"
                    count++;
    }
    return count;
}

int count_article_by_splitting(char *str) {
    int count = 0, len = strlen(str) - 1, word_count = 0;
    char *token[WORD_SIZE], currentchar[2];
    currentchar[1] = '\0';  // to append characters to token one by one

    // allocate memory
    for (int i = 0; i < WORD_SIZE; i++) {
        token[i] = malloc (WORD_SIZE * sizeof(char));
    }

    // can also use strtok() string method for the same purpose
    // tokenize and store the words separated by space, comma, or period
    /*
    1. Initialize the token array with the number of words in the string.
    2. Initialize the currentchar array with the first character of the string.
    3. Loop through the string, if the current character is not a space, comma, or period, add
it to
    the currentchar array.
    4. If the current character is a space, comma, or period, add the currentchar array to the
token
    array and increment the word_count.
    5. Return the token array.
    */
    for (int i = 0; i <= len; i++) {
        if (str[i] != ' ' && str[i] != ',' && str[i] != '.') {
        // if (str[i] < 'a' || str[i] > 'Z') {  // use this to assume any non alpha character as
a separator
            currentchar[0] = str[i];
            strcat(token[word_count], currentchar);
        } else {
            word_count++;
        }
    }
    for (int i = 0; i <= word_count; i++) {
        if (strcmp(token[i], "a") == 0 || strcmp(token[i], "an") == 0 || strcmp(token[i], "the")
== 0) {
            count++;
        }

    }

    return count;
}

/*
Convert all characters in a string to lowercase.
*/
int str_tolower(char *str) {
    for (int i = 0; i < strlen(str); i++) {
```

```
        str[i] = tolower(str[i]);
    }
    return 0;
}
```



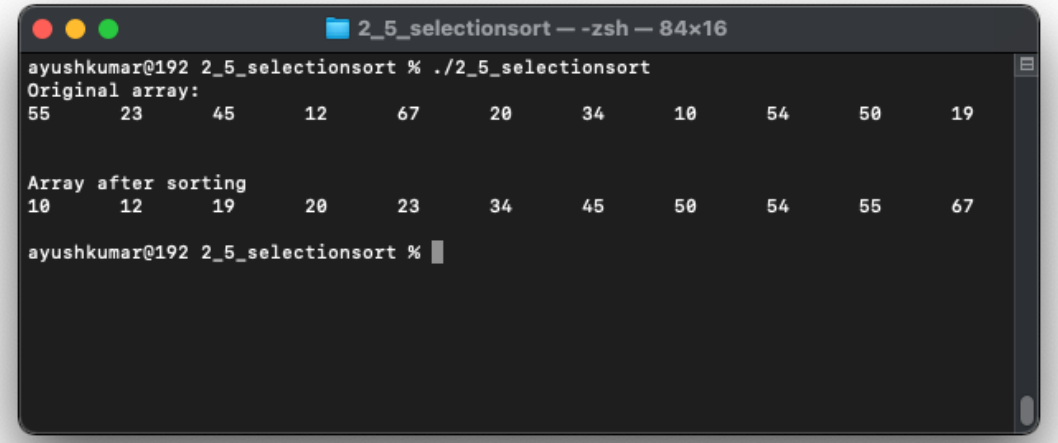
Section 2 (Sorting and Searching)

5. Execute Selection sort algorithm **manually** to sort the following numbers

   55, 23, 45, 12, 67, 20, 34, 10, 54, 50, 19

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

#define ARR_SIZE 11

void selection_sort(int arr[], int size);

int main() {
    int set[] = {55, 23, 45, 12, 67, 20, 34, 10, 54, 50, 19}, pos;

    printf("Original array:\n");
    for (int i = 0; i < ARR_SIZE; i++) printf("%d\t", set[i]);

    selection_sort(set, ARR_SIZE);
    printf("\n\n");
    printf("Array after sorting\n");
    for (int i = 0; i < ARR_SIZE; i++) printf("%d\t", set[i]);

    printf("\n");
    return 0;
}

/* "The selection sort algorithm sorts an array by repeatedly finding the minimum element
(considering ascending order) from unsorted part and putting it at the beginning."

The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the
unsorted subarray is picked and moved to the sorted subarray.
```

```
The idea of the algorithm is to go through the array and find the minimum */
void selection_sort(int arr[], int size) {
    int temp;
    for (int i = 0; i < size - 1; i++) {
       for (int j = i + 1; j < size; j++) {
           if (arr[i] > arr[j]) {
               temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
            }
        }
   }
}
```



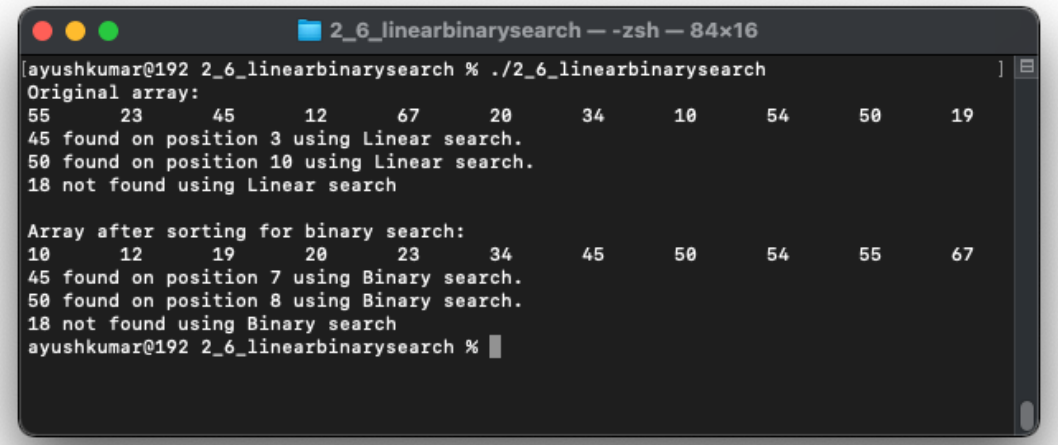
6. How do you search the elements 45, 50 and 18 with the Linear and Binary search methods?

Assume that the following set of elements are given:
55, 23, 45, 12, 67, 20, 34, 10, 54, 50, 19

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

#define ARR_SIZE 11

void sort(int arr[], int size);
int linear_search(int[], int, int);
int binary_search(int[], int, int);

int main() {
    int set[] = {55, 23, 45, 12, 67, 20, 34, 10, 54, 50, 19}, pos;

    printf("Original array:\n");
    for (int i = 0; i < ARR_SIZE; i++) printf("%d\t", set[i]);

    pos = linear_search(set, 45, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Linear search.", 45, pos); else
printf("\n%d not found using Linear search", 45);
    pos = linear_search(set, 50, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Linear search.", 50, pos); else
printf("\n%d not found using Linear search", 50);
```

```c
    pos = linear_search(set, 18, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Linear search.", 18, pos); else
printf("\n%d not found using Linear search", 18);

    sort(set, ARR_SIZE);
    printf("\n\n");
    printf("Array after sorting for binary search:\n");
    for (int i = 0; i < ARR_SIZE; i++) printf("%d\t", set[i]);

    pos = binary_search(set, 45, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Binary search.", 45, pos); else
printf("\n%d not found using Binary search", 45);
    pos = binary_search(set, 50, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Binary search.", 50, pos); else
printf("\n%d not found using Binary search", 50);
    pos = binary_search(set, 18, ARR_SIZE);
    if (pos != -1) printf("\n%d found on position %d using Binary search.", 18, pos); else
printf("\n%d not found using Binary search", 18);


    printf("\n");
    return 0;
}

/* Given an array of integers, find the position of a given integer in the array. */
int linear_search(int arr[], int element, int size) {
    int pos = -2;
    for (int i = 0; i < size; i++) {
        if (arr[i] == element) {
            pos = i;
            break;
        }
    }
    return pos+1;
}

/* Given an array of integers, sort the array in ascending order. */
void sort(int arr[], int size) {
    int temp;
    for (int i = 0; i < size - 1; i++) {
       for (int j = i + 1; j < size; j++) {
           if (arr[i] > arr[j]) {
               temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
            }
        }
   }
}

/* Given an array of integers, find the index of the element that is equal to the given integer.
*/
int binary_search(int arr[],int element, int size) {
  int f = 0, r = size, mid;

   while (f <= r) {
       mid = (f + r) / 2;
```

```
        if (arr[mid] == element) {
            return mid + 1;
            break;
        }
        else if (arr[mid] < element)
            f = mid + 1;
        else
            r = mid - 1;
    }
    if (f > r)
        return -1;
     return -2;
}
```



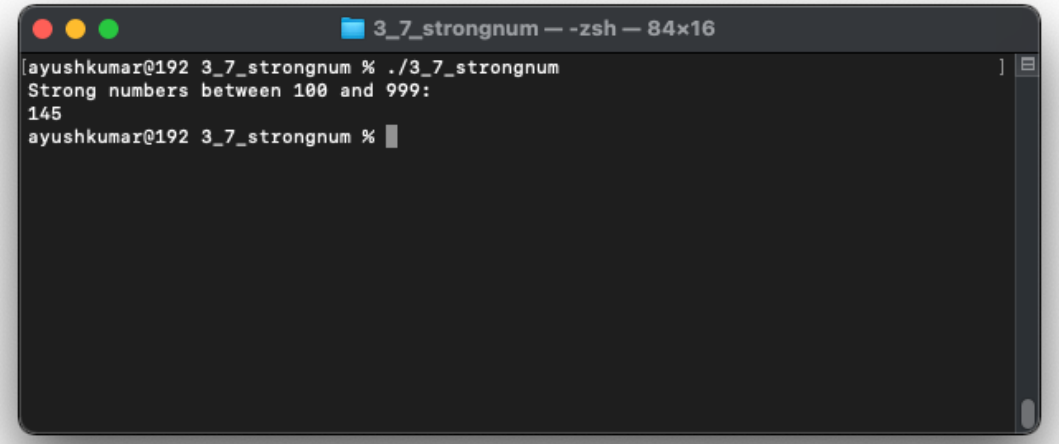
Section 3 (Loops and conditions)

7. Write a program to display all the strong numbers between 100 to 999

A strong number is a number in which the sum of the factorial of the digits is equal to the number itself. (Example: 145 = 1! + 4! + 5!)

```
#include <stdio.h>

int check_strong_num(int);

int main(int argc, char const *argv[]) {
    printf("Strong numbers between 100 and 999:\n");
    for (int i = 100; i < 1000; i++)
        if (check_strong_num(i) == 0) printf("%d\t", i);

    printf("\n");
    return 0;
}

/*
1. Get the last digit of the input number.
    2. Multiply all the digits from 1 to the last digit.
    3. Add the result to the fact_sum.
    4. Repeat steps 1-3 until the input number is 0.
    5. If the fact_sum is equal to the input number, return 0. Strong number
    6. Else return 1. Not a strong number
```

```c
*/
int check_strong_num(int input) {
    int fact_sum = 0, num = input;
    while (num > 0) {
        int digit = num % 10;   // get the last digit
        num /= 10;  // remove last digit
        int fact = 1;
        for (int j = digit; j > 1; j--) {   // j > 1 because there is no point multiplying with
1 every time
            fact *= j;
        }
        fact_sum += fact;
    }
    if (fact_sum == input) return 0;
    else return 1;
}
```

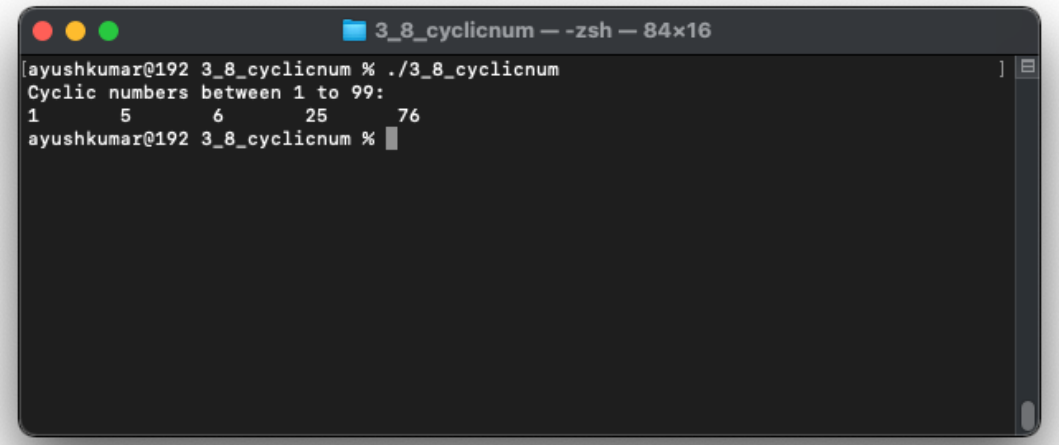


8. Write a program to display all the cyclic numbers between 1 to 99.

A cyclic number is a number if its square ends with the number. Examples: 6 ($6^2 = 36$), 5 ($5^2 = 25$) ,76 ($76^2 = 5776$)

```c
#include <stdio.h>
#include <math.h>

int check_cyclic_num(int);

int main(int argc, char const *argv[]) {
    printf("Cyclic numbers between 1 to 99:\n");
    for (int i = 1; i < 100; i++)
        if (check_cyclic_num(i) == 0) printf("%d\t", i);

    printf("\n");
    return 0;
}

int check_cyclic_num(int input) {
    int square = input * input, num = input, flag = 0;

    /*
    If the last digit of the number is not the same as the digits of the square, then the number
```

```
    is not a cyclic number.
    */
    for (int i = 1; num > 0; i++) {
        if (square % 10 != num % 10) {
            return 1;
        }
        num /= 10;
        square /= 10;
    }
    return 0;
}
```



Section 4 (Pointers and Structures)

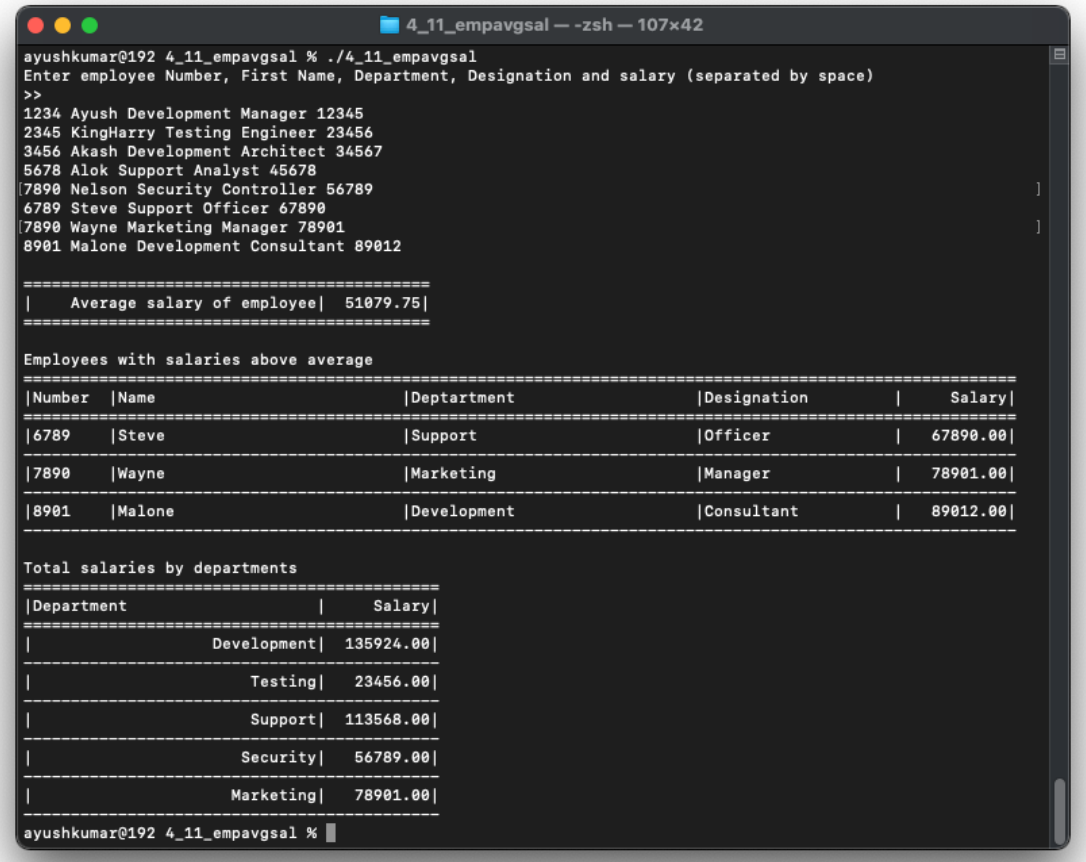
11. Assume that there are ‘n’ employee records. Each employee record consists of employee name, employee number, department, designation and salary. Write a program to
    a. Find the average salary of the employees
    b. Display all the employees who are getting more than average salary
    c. Find the total salary drawn by employees in each department

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

#define OBJ_SIZE 8
#define SEP_LINE_LENGTH 105

struct employee {
    char num[8], name[30], dept[30], desig[20];
    double salary;
};

double calc_emp_avg_salary(struct employee[]);
double calc_total_dept_salary(char*, struct employee[]);
int get_unique_dept(char[OBJ_SIZE][30], struct employee[]);
void emp_input(struct employee[], int);
void emp_display(struct employee);
void print_header();
void print_dept_header();
```

```c
int main(int argc, char const *argv[]) {
    struct employee emp[OBJ_SIZE];

    emp_input(emp, OBJ_SIZE);

    // Print average salary of the employees
    printf("\n");
    for (int i = 0; i < 43; i++) printf("=");
    printf("\n|%30s|%10.2lf|\n", "Average salary of employee", calc_emp_avg_salary(emp));
    for (int i = 0; i < 43; i++) printf("=");
    printf("\n");

    // print employees with salaries higher than average
    printf("\nEmployees with salaries above average");
    print_header();
    /*
    The code displays the employees whose salary is greater than the average salary.
    */
    for (int i = 0; i < OBJ_SIZE; i++) {
        if (emp[i].salary > calc_emp_avg_salary(emp)) {
            emp_display(emp[i]);
        }
    }

    // make a list of unique departments
    char dept[OBJ_SIZE][30];
    int dept_count;
    dept_count = get_unique_dept(dept, emp);
    // calculate total salary of departments
    printf("\nTotal salaries by departments");
    print_dept_header();
    /*
    Print a table of the total salary for each department.
    */
    for (int i = 0; i < dept_count; i++) {
        printf("\n|%30s|%11.2lf|\n", dept[i], calc_total_dept_salary(dept[i], emp));
        for (int i = 0; i < 44; i++) printf("-");
    }



    // for (int i = 0; i < OBJ_SIZE; i++) emp_display(emp[i]);

    printf("\n");
    return 0;
}

double calc_emp_avg_salary(struct employee obj[]) {
    double avg_salary;
    /*
    Calculating the average salary of all the objects in the array of employees.
    */
    for (int i = 0; i < OBJ_SIZE; i++) {
        avg_salary += obj[i].salary;
    }
    avg_salary /= OBJ_SIZE;
    return avg_salary;
```

```c
}

double calc_total_dept_salary(char *dept, struct employee obj[]) {
    double curr_salary_sum = 0;
   /*
   For each department, add up the salaries of all employees in that department.
   */
    for (int j = 0; j < OBJ_SIZE; j++) {
          if (strcmp(dept, obj[j].dept) == 0) curr_salary_sum += obj[j].salary;
    }
    return curr_salary_sum;
}

int get_unique_dept(char dept[OBJ_SIZE][30], struct employee obj[]) {
    int count = 0;
    /*
    1. For each object in the array, check if the department name is already in the dept array.
    2. If it is not, add it to the dept array.
    3. Increment the count variable.
    */
    for (int i = 0; i < OBJ_SIZE; i++) {
        int flag = 1;
        for (int j = 0; j <= i; j++) {
            if (strcmp(dept[j], obj[i].dept) == 0) {
                flag = 0;
                break;
            }
        }
        if (flag == 1) {
            strcpy(dept[count], obj[i].dept);
            count++;
        }
    }
    return count;
}

/*
The function emp_input() takes an array of employee objects and a count of the number of objects
as
input. It then prompts the user to enter the employee details and stores them in the employee
objects.
*/
void emp_input(struct employee obj[], int count) {
    printf("Enter employee Number, First Name, Department, Designation and salary (separated by
space)\n");
    printf(">> \n");
    for (int i = 0; i < count; i++) {
        scanf("%s%s%s%s%lf", obj[i].num, obj[i].name, obj[i].dept, obj[i].desig,
&obj[i].salary);
    }
}

/*
The function emp_display() takes a struct employee object as an argument and displays the
employee
details.
*/
```

```c
void emp_display(struct employee obj) {
    printf("|%-8s|%-30s|%-30s|%-20s|%11.2lf|\n", obj.num, obj.name, obj.dept, obj.desig,
obj.salary);
    for (int i = 0; i < SEP_LINE_LENGTH; i++) printf("-"); printf("\n");
}

/*
Prints a header for the employee table.
*/
void print_header() {
    printf("\n");
    for (int i = 0; i < SEP_LINE_LENGTH; i++) printf("="); printf("\n");
    printf("|%-8s|%-30s|%-30s|%-20s|%11s|\n", "Number", "Name", "Deptartment", "Designation",
"Salary");
    for (int i = 0; i < SEP_LINE_LENGTH; i++) printf("="); printf("\n");
}

/*
Prints a header for the department report.
*/
void print_dept_header() {
    printf("\n");
    for (int i = 0; i < 44; i++) printf("=");
    printf("\n|%-30s|%11s|\n", "Department", "Salary");
    for (int i = 0; i < 44; i++) printf("=");
}
```

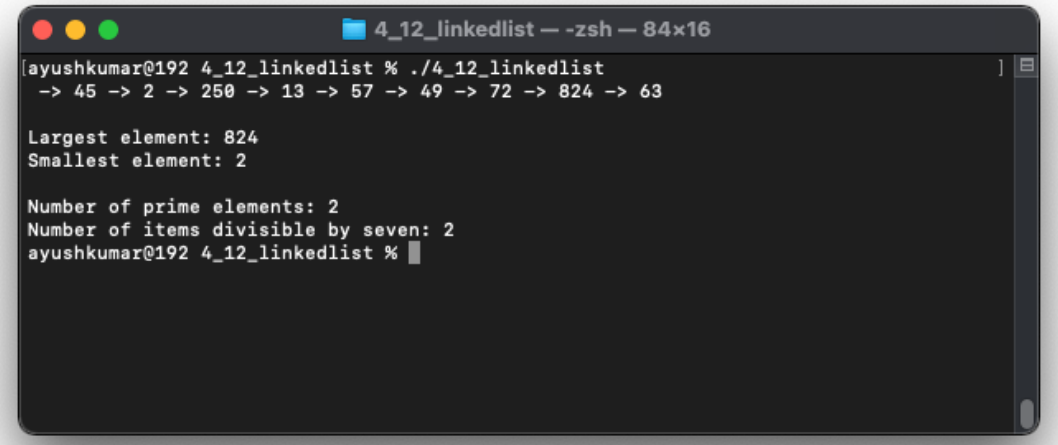


12. Assume that there is a linked list that begins with Begin and ends with End pointers. Write a program to
    a. Find the biggest and smallest elements in the list
    b. Find the number of elements which are prime
    c. Find the number of elements which are divisible by 7

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

/*
Creating a linked list.
*/
struct item {
    int data;
    struct item *link;
} *first = NULL,    // begin
  *cursor = NULL;   // end

struct item* insert_item(int);
struct item* find_largest();
struct item* find_smallest();
int count_prime_items();
int count_items_div_seven();
void display_list();

int main(int argc, char const *argv[]) {
    // can use scanf for dynamic insertion of items
    insert_item(45);
    insert_item(2);
    insert_item(250);
    insert_item(13);
    insert_item(57);
    insert_item(49);
    insert_item(72);
    insert_item(824);
    insert_item(63);

    display_list();

    printf("\n\nLargest element: %d", find_largest()->data);
    printf("\nSmallest element: %d\n", find_smallest()->data);

    printf("\nNumber of prime elements: %d", count_prime_items());
    printf("\nNumber of items divisible by seven: %d", count_items_div_seven());

    printf("\n");
    return 0;
}

struct item* insert_item(int data) {
    struct item *node = (struct item*) malloc(sizeof(struct item));
    struct item *temp;

    node->data = data;
    node->link = NULL;
```

```
    /*
    If the list is empty,
    assign the node to the first pointer. If the list has only one node,
    assign the node to the cursor pointer. If the cursor pointer is null,
    assign the node to the cursor pointer. Otherwise, assign the node to the
    cursor link item.
    */
    if (first == NULL) {  // if the list is empty
        first = node;
        cursor = first;
    } else if (first->link == NULL) {   // if the list has only one item
        first->link = node;
        cursor = node;
    } else if (cursor->link == NULL) {  // if the cursor->link is null,
                                        // and we try to assign that address to cursor, it will
fail
        cursor->link = node;
    } else {
        cursor = cursor->link;
        cursor->link = node;
    }

    return node;
}

struct item* find_largest() {
    struct item *ptr = first;
    struct item *large = (struct item*) malloc(sizeof(struct item));

    large->data = INT_MIN;  // initialize with smallest possible number

    /*
    Finding the largest value in the list.
    */
    while (ptr != NULL) {
        if (ptr->data > large->data) large = ptr;
        ptr = ptr->link;
    }

    return large;
}

struct item* find_smallest() {
    struct item *ptr = first;
    struct item *small = (struct item*) malloc(sizeof(struct item));

    small->data = INT_MAX;  // initialize with smallest possible number

   /*
   Finding the smallest element in the list.
   */
    while (ptr != NULL) {
        if (ptr->data < small->data) small = ptr;
        ptr = ptr->link;
    }
```

```c
    return small;
}

int count_prime_items() {
    struct item *ptr = first;
    int count = 0;
    /*
    Count the number of prime numbers in a linked list.
    */
    while (ptr != NULL) {
        int flag = 0;
        for (int i = 2; i <= (ptr->data)/2; i++)
            if (ptr->data % i == 0) flag = 1;

        if (flag == 0) count++;

        ptr = ptr->link;
    }
    return count;
}

int count_items_div_seven() {
    struct item *ptr = first;
    int count = 0;
    /*
    Counting the number of elements in the list that are divisible by 7.
    */
    while (ptr != NULL) {
        if (ptr->data % 7 == 0) count++;
        ptr = ptr->link;
    }
    return count;
}

void display_list() {
    struct item *node = first;
    /*
    Printing the data of each node in the linked list.
    */
    while (node != NULL) {
        printf(" -> %d", node->data);
        node = node->link;
    }
}
```



Section 5 (Arrays and Files)

13. Write a C program to define a structure called student with data members, name, register number, major, marks in core subject1, core subject2, allied, elective, total, average and grade. The data for name, register number, major, core subject1, core subject2, allied and elective are to be obtained from user.
    a. Calculate total, average and grade for the students and update the data in the file.
    b. Search for a student whose register number is obtained from the user as console input.
    c. Find the first three highest averages and print those student's detail

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

#define TABLE_LINE_LENGTH 110
#define FILE_PATH "studata.dat"

/* The struct student is a data structure that contains the following data:

- msub1: marks of subject 1
- msub2: marks of subject 2
- elective: marks of elective subject
- allied: marks of allied subject
- avg: average marks
- total: total marks
- regno: registration number
- grade: grade
- name: name of the student
- major: major of the student */
struct student {
    double msub1, msub2, elective, allied, avg, total;
    char regno[10], grade, name[20], major[20];
};

int get_student_input(struct student*);
int get_student_batch_input(int);
void display_student_details(struct student);
int search_student(char*);
```

```c
void calc_scores(struct student*);
int print_highest_three_avg();
int insert_data_to_file(struct student);
void print_header();
int display_all_student_data();

int main(int argc, char const *argv[]) {
    struct student obj;
    int opt = 0;

    /* The main function is responsible for taking user input and calling the appropriate
functions. */
    printf("Enter a choice:\n1. Add new student data\n2. Batch input student data\n3. Display
all student data\n4. Search for student\n5. Display students with top 3 averages\n6. Exit");
    while (opt != 6) {
        printf("\nMain>> ");
        scanf("%d", &opt);
        // opt = 5;

        switch (opt) {
            case 1: {   // Add new student data
                get_student_input(&obj);
                calc_scores(&obj);
                insert_data_to_file(obj);
                break;
            } case 2: {  // Batch input student data
                int count;
                printf("Enter number of records: ");
                scanf("%d", &count);
                printf("Enter Register Number, First Name, Major, 1st core subject marks(100),
2nd core subject marks(100), allied marks(100) and elective marks(100) (separated by space, new
entries on new line)\nBatch Input Below>>\n");
                get_student_batch_input(count);
                break;
            } case 3: { // Display all student data
                printf("\nDisplay all student data\n");
                display_all_student_data();
                break;
            } case 4: { // Search student
                char regno[10];
                printf("Enter the register number to search\nSearch student>> ");
                scanf("%s", regno);
                print_header();
                search_student(regno);
                break;
            } case 5: { // Display students with highest 3 averages
                printf("\nDisplay students with highest 3 averages\n");
                print_header();
                print_highest_three_avg();
                break;
            } case 6: { // Exiting
                printf("\nExiting...\n");
                break;
            } default: {    //Invalid choice
                printf("\nInvalid choice, please try again.");
                break;
            }
```

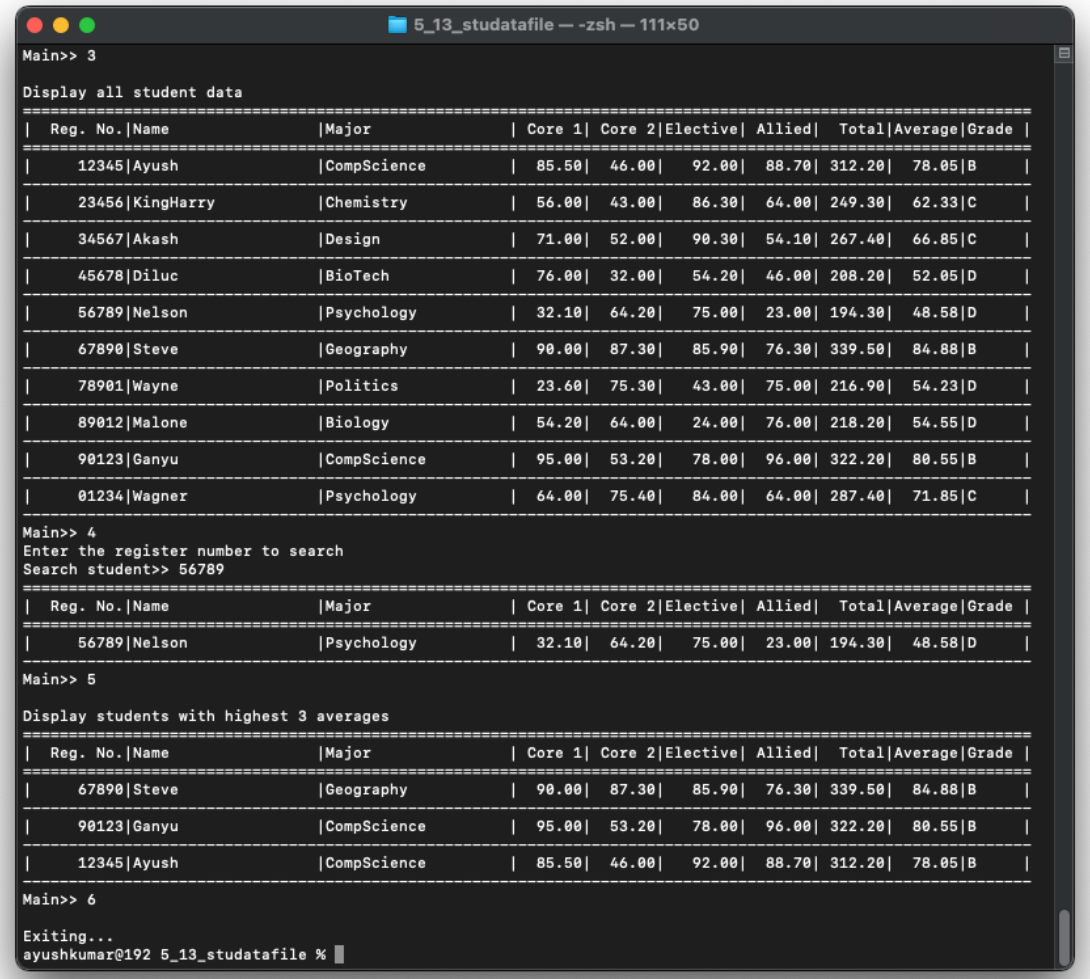
```
        }
    }
    return 0;
}

/* The function get_student_input() takes a pointer to a struct student as an argument. It then
asks
the user to enter the student's details and stores them in the struct student.

The function get_student_input() is called in the main() function.*/
int get_student_input(struct student *obj) {
    printf("Enter Register Number(10 digit), First Name, Major, 1st core subject marks(100), 2nd
core subject marks(100), allied marks(100) and elective marks(100) (separated by space)\nNew
Student>> ");
    scanf("%s%s%s%lf%lf%lf%lf", obj->regno, obj->name, obj->major, &obj->msub1, &obj->msub2,
&obj->elective, &obj->allied);
    if (obj->msub1 > 100 || obj->msub2 > 100 || obj->elective > 100 || obj->allied > 100) {
        printf("\nError in marks input (max. 100).\n"); return 1;
    }
    return 0;
}

/* The function get_student_batch_input() takes in the number of students to be entered as an
argument. It then prompts the user to enter the details of each student. The details are then
stored
in a struct student object. The function calc_scores() is called to calculate the scores of each
student. The function insert_data_to_file() is called to write the details of each student to
the
file. The function get_student_batch_input() returns 0 if all the students are successfully
written
to the file. */
int get_student_batch_input(int count) {
    struct student obj;
    int successwritecount = 0, failwritecount = 0;
    char *failedregnos[count];
    printf("================================================\n");
    for (int i = 0; i < count; i++) {
        scanf("%s%s%s%lf%lf%lf%lf", obj.regno, obj.name, obj.major, &obj.msub1, &obj.msub2,
&obj.elective, &obj.allied);
        if (obj.msub1 > 100 || obj.msub2 > 100 || obj.elective > 100 || obj.allied > 100) {
            printf("\nError in marks input (max. 100).\n"); return 1;
        }
        calc_scores(&obj);
        if (insert_data_to_file(obj) == 0) {
            successwritecount++;
        } else {
            strcpy(failedregnos[failwritecount], obj.regno);
            failwritecount++;
        }
    }
    printf("================================================");
    printf("\n%d records written successfully.\n%d records failed to write.", successwritecount,
failwritecount);
    if (failwritecount > 0) {
        printf("\nThe records with following reg.no.(s) failed to write.");
        /* Printing the failedregnos array. */
        for (int i = 0; i < failwritecount; i++)
```

```
            printf("\n%s", failedregnos[i]);
    }
    return 0;
}

/* Calculate the scores for each student and assign the grade. */
void calc_scores(struct student *obj) {
    obj->total = obj->msub1 + obj->msub2 + obj->elective + obj->allied;
    obj->avg = obj->total / 4;

    /* If the average is greater than 85, assign the letter grade A.
    If the average is greater than 75, assign the letter grade B.
    If the average is greater than 60, assign the letter grade C
    If the average is greater than 45, assign the letter grade D.
    If the average is less than or equal to 35, assign the letter grade F.*/
    if (obj->avg > 85) obj->grade = 'A';
    else if (obj->avg > 75) obj->grade = 'B';
    else if (obj->avg > 60) obj->grade = 'C';
    else if (obj->avg > 45) obj->grade = 'D';
    else if (obj->avg <= 35) obj->grade = 'F';
}

/* Write the object to the file. */
int insert_data_to_file(struct student obj) {
    FILE *file;
    file = fopen(FILE_PATH, "a");
    if (file == NULL) {printf("\nError opening file.\n"); return 1;}

    /* Write the object to the file. */
    int write_rc = fwrite(&obj, sizeof(struct student), 1, file);
    if (write_rc <= 0) {printf("\nError writing to file.\n"); return 2;}

    fclose(file);
    return 0;
}

/* Display the student details. */
void display_student_details(struct student obj) {
    printf("\n|%10s|%-20s|%-20s", obj.regno, obj.name, obj.major);
    printf("|%7.2lf|%7.2lf|%8.2lf|%7.2lf", obj.msub1, obj.msub2, obj.elective, obj.allied);
    printf("|%7.2lf|%7.2lf|%-6c|\n", obj.total, obj.avg, obj.grade);
    for (int i = 0; i < TABLE_LINE_LENGTH; i++) printf("-");
}

/* Print a header for the table. */
void print_header() {
    for (int i = 0; i < TABLE_LINE_LENGTH; i++) printf("=");
    printf("\n|%10s|%-20s|%-20s", "Reg. No.", "Name", "Major");
    printf("|%7s|%7s|%8s|%7s", "Core 1", "Core 2", "Elective", "Allied");
    printf("|%7s|%7s|%-6s|\n", "Total","Average", "Grade");
    for (int i = 0; i < TABLE_LINE_LENGTH; i++) printf("=");
}

/* Display all student data from the file. */
int display_all_student_data() {
    struct student obj;
    FILE *file;
```

```c
    file = fopen(FILE_PATH, "r");
    if (file == NULL) {printf("\nError opening file.\n"); return 1;}

    print_header();
    /* Reading the file and displaying the details of the student. */
    while (fread(&obj, sizeof(struct student), 1, file)) {
        display_student_details(obj);
    }
    fclose(file);
    return 0;
}

/* The search_student() function searches the file for a student with the given registration
number.
If found, the function displays the student's details.

The search_student() function returns 0 if the search was successful, and 1 if the search
failed.*/
int search_student(char regno[]) {
    FILE *file;
    int count = 0;
    struct student obj;
    file = fopen(FILE_PATH, "r");
    if (file == NULL) {printf("\nError opening file.\n"); return 1;}

    while (fread(&obj, sizeof(struct student), 1, file)) {
        if (strcmp(obj.regno, regno) == 0) {
            display_student_details(obj);
            count++;
        }
    }
    if (count <= 0) printf("No matching records found");
    fclose(file);
    return 0;
}

/* The function searches for the highest three averages and prints them out.*/
int print_highest_three_avg() {
    struct student obj;
    FILE *file;
    double highestavgs[] = {INT_MIN, INT_MIN, INT_MIN}, found = 1;
    char highavgregno[3][10];
    file = fopen(FILE_PATH, "r");
    if (file == NULL) {printf("\nError opening file.\n"); return 1;}

    /* The highestavgs array is initialized to all 0's.
    The for loop iterates through the array of highest avgs.

    If the current object's average is greater than the current highest average,
    the highest average is set to the current object's average.
    The highest average is then stored in the highestavgs array.

    The regno of the object with the highest average is stored in the highavgregno
    array. Later this regno will be used to search for and print the students with
    highest averages.

    The for loop then iterates to the next object in the array. */
```

```c
    for (int i = 0; i < 3; i++) {
        rewind(file);
        /* If the average of the student is larger than the current largest average, then save
the
        average and the student's registration number. */
        while (fread(&obj, sizeof(struct student), 1, file)) {
            if (obj.avg > highestavgs[i]) { // if larger than current largest saved
                found = 1;
                for (int j = 0; j < 3; j++) {
                    if (highestavgs[j] == obj.avg) {    // if other highest is same as this one
                        found = 0;
                    }
                }
                /* Checking if the user has already been found in the database. */
                if (found == 1) {
                    highestavgs[i] = obj.avg;
                    strcpy(highavgregno[i], obj.regno);
                }
            }
        }
    }
    fclose(file);
    /* Searching for the student with the highest average grade. */
    for (int i = 0; i < 3; i++) {
        search_student(highavgregno[i]);
    }
    return 0;
}
```

```
5_13_studatafile — -zsh — 111×39
ayushkumar@192 5_13_studatafile % ./5_13_studatafile
Enter a choice:
1. Add new student data
2. Batch input student data
3. Display all student data
4. Search for student
5. Display students with top 3 averages
6. Exit
Main>> 1
Enter Register Number(10 digit), First Name, Major, 1st core subject marks(100), 2nd core subject marks(100), a
llied marks(100) and elective marks(100) (separated by space)
New Student>> 12345 Ayush CompScience 85.5 46 92 88.7

Main>> 3

Display all student data
=============================================================================================================
|  Reg. No.|Name               |Major              | Core 1| Core 2|Elective| Allied|  Total|Average|Grade |
=============================================================================================================
|     12345|Ayush              |CompScience        |  85.50|  46.00|   92.00|  88.70| 312.20|  78.05|B     |
-------------------------------------------------------------------------------------------------------------
Main>> 2
Enter number of records: 9
Enter Register Number, First Name, Major, 1st core subject marks(100), 2nd core subject marks(100), allied mark
s(100) and elective marks(100) (separated by space, new entries on new line)
Batch Input Below>>
=============================================
23456 KingHarry Chemistry 56 43 86.3 64
34567 Akash Design 71 52 90.3 54.1
45678 Diluc BioTech 76 32 54.2 46
56789 Nelson Psychology 32.1 64.2 75 23
67890 Steve Geography 90 87.3 85.9 76.3
78901 Wayne Politics 23.6 75.3 43 75
89012 Malone Biology 54.2 64 24 76
90123 Ganyu CompScience 95 53.2 78 96
01234 Wagner Psychology 64 75.4 84 64
=============================================
9 records written successfully.
0 records failed to write.
```



14. Assume that there is an array consisting of ‘n’ elements. Write a program to store all the prime numbers of the list in a file called as “PRIME” and non-prime numbers in another file called as “NPRIME”. You have to define a function to check whether an element is a prime or not

```c
#include <stdio.h>

#define ARR_SIZE 10

int isPrime(int);

int main(int argc, char const *argv[]) {
    int input[ARR_SIZE], prime[ARR_SIZE], nprime[ARR_SIZE], primectr = 0, nprimectr = 0;

    printf("Enter the array elements (separated by space)\n>> ");
    for (int i = 0; i < ARR_SIZE; i++) scanf("%d", &input[i]);
    /* this process should be done as soon as taking input in the same loop,
       to avoid creating another array for storing the numbers and also
       avoid the use of another loop */
    for (int i = 0; i < ARR_SIZE; i++) {
        if (isPrime(input[i]) == 0) {
            prime[primectr] = input[i];
            primectr++;
        } else {
            nprime[nprimectr] = input[i];
            nprimectr++;
```

```c
        }
    }

    printf("\nPrime elements:\n");
    for (int i = 0; i < primectr; i++)  printf("%d\t", prime[i]);
    printf("\nNon Prime elements:\n");
    for (int i = 0; i < nprimectr; i++) printf("%d\t", nprime[i]);

    printf("\n");
    return 0;
}

/*
Given an integer n, return 1 if n is prime, 0 otherwise.
*/
int isPrime(int n) {
    int flag = 0;
    for (int i = 2; i <= n/2; i++) {
        if (n % i == 0) {
            flag = 1;
            break;
        }
    }
    return flag;
}
```