

## Practice – 10

## Pointers

## (a) Pointers in C

Objective: learn to implement the basic functionalities of pointers in C.

<https://www.hackerrank.com/challenges/pointer-in-c/problem?isFullScreen=true>

```
#include <stdio.h>
#include <math.h> void
update(int *a,int *b) {
    int t = *a;
    *a = *a + *b;
    *b = abs(t - *b);
}

int main() {
    int a, b;
    int *pa = &a, *pb = &b;
    scanf("%d %d", &a, &b);
    update(pa, pb);    printf("%d\n%d",
a, b);
    return 0; }
```

Output:

15 7  
22 8

(b)  
**Stu  
den  
ts  
Ma  
rks  
Su  
m**

Objective: Learn using Pointers with Arrays and Functions

<https://www.hackerrank.com/challenges/students-marks-sum/problem?isFullScreen=true>

```
#include <stdio.h>
#include <string.h>
#include <math.h> #include
<stdlib.h>
int marks_summation(int* marks, int number_of_students, char gender) {
    int sum = 0;
```

```
    for (int i = gender == 'b' ? 0 : 1; i < number_of_students; i += 2) {  
        sum += marks[i];  
    }  
    return sum;  
} int main()  
{  
    int number_of_students;  
    char gender;  
    int sum;  
    printf("Enter Number of Students: ");  
    scanf("%d", &number_of_students);  
    int *marks = (int *) malloc(number_of_students * sizeof(int));  
    printf("Enter marks: ");  
    for (int student = 0; student < number_of_students; student++) {  
        scanf("%d", (marks + student));  
    }  
    printf("Enter Gender: ");  
    scanf(" %c", &gender);  
    sum = marks_summation(marks, number_of_students, gender);  
    printf("%d", sum);  free(marks);  return 0; }
```

**Output 1:**

Enter Number of Students: 3  
Enter marks: 3 2 5  
Enter Gender: b 8

**Output 2:**

Enter Number of Students: 6  
Enter marks: 24 32 16 4 7 9  
Enter Gender: g 45

---

---

**Roll Number:**

**Page: 56**

### (c) Sorting Array of Strings

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define size_of_array 3 //define size of the array
//function to display array void
display(char array[][30])
{
    for(int i=0; i<size_of_array; i++)
    {
        printf("%s ", array[i]);
    }
    printf("\n");
}
int main(){
    //create an array of strings
    char array[size_of_array][30];
    //Inputting names
    printf("Enter %d Strings: \n", size_of_array);
    for(int i=0; i<size_of_array; i++){
        scanf("%s", array[i]);
    }
    //display the original array
    printf("Original array: ");
    display(array); char
    temp[30];
    //Sort array using the Buuble Sort
    algorithm for(int i=0; i<size_of_array;
    i++){ for(int j=0; j<size_of_array-1-i;
    j++){ if(strcmp(array[j], array[j+1]) >
    0){ //swap array[j] and array[j+1]
    strcpy(temp, array[j]); strcpy(array[j],
    array[j+1]);
    strcpy(array[j+1], temp);
    }
    }
    }
    //display the sorted
    array printf("Sorted
    Array: ");
    display(array); return 0;
}
```

#### Output:

Enter 3 Strings:  
Ball Apple Cat  
Original array: Ball Apple Cat

Sorted Array: Apple Ball Cat

---

---

**Roll Number:**

**Page: 57**

**(d) Find the sum of a 1D array using malloc()**

```
#include <stdio.h>
#include <stdlib.h>
int
main()
{
    int size;
    printf("Enter size of the array: ");
    scanf("%d", &size);
    // Allocating memory dynamically using malloc
    int *array = (int *)malloc(size * sizeof(int)); //
    Reading values and finding out the sum    int sum
    = 0, i;
    printf("Enter %d array elements: ", size);
    for (i = 0; i < size; i++)
    {
        scanf("%d", array + i); // same as writing &arr[i];
        sum = sum + *(array + i); // same as writing arr[i];
    }
    printf("Sum is: %d", sum);
    return 0; }
```

**Output:**

```
Enter size of the array: 5
Enter 5 array elements: 10 20 30 40 50 Sum
is: 150
```

## **(f) Swap two numbers using functions and pointers - call by value and reference**

### **(i) call by reference**

```
#include <stdio.h> void
swap (int *, int *); int
main(){
    int a, b;
    printf("\nEnter value of a & b: ");
    scanf("%d %d", &a, &b);
    printf("\nBefore Swapping:\n");
    printf("a = %d\nb = %d\n", a, b);
    swap(&a, &b);    printf("\nAfter
Swapping:\n");    printf("a =
%d\nb = %d", a, b);
    return 0; }
void swap (int *x, int *y)
{    int temp;
temp = *x;
    *x = *y;
    *y = temp;
}
```

### **Output:**

```
Enter value of a & b: 10 20 Before
Swapping:
a = 10 b = 20
After Swapping:
a = 20
b = 10
```

**(ii) call by value**

```
#include<stdio.h>
void swap (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("After swapping 1st number is %d and 2nd number is %d", a ,b);
}
int main(void)
{
    int first, second;
    //printf("Enter two numbers : \n");
    scanf("%d %d",&first,&second);
    swap(first,second);
    printf("\n After swap function called 1st number is %d and 2nd number is %d", first
,second);
    return 0;
}
```

**Output:**

3  
7

After swapping 1st number is 7 and 2nd number is 3

After swap function called 1st number is 3 and 2nd number is 7

## (g) Dynamic Array in C

Objective: Handling requests by a Librarian to place the books in the shelves.

<https://www.hackerrank.com/challenges/dynamic-array-inc/problem?isFullScreen=true>

```
#include <stdio.h>
#include <stdlib.h> /*
 * This stores the total number of books in each shelf.
 */
int* total_number_of_books;
/*
 * This stores the total number of pages in each book of each shelf.
 * The rows represent the shelves and the columns represent the books.
 */
int** total_number_of_pages;
int main() {
    int total_number_of_shelves;    scanf("%d", &total_number_of_shelves);
    total_number_of_books = calloc (total_number_of_shelves, sizeof(int));
    total_number_of_pages = calloc (sizeof(int *), total_number_of_shelves);
    int total_number_of_queries;
    scanf("%d", &total_number_of_queries);

    while (total_number_of_queries--) {
        int type_of_query;    scanf("%d",
        &type_of_query);
        if (type_of_query == 1) {
            /*
             * Process the query of first type here.
             */
            int x, y;
            scanf("%d %d", &x, &y);

            // update number of books in the shelf [x]
            total_number_of_books[x]++;

            // insert book at the end of the shelf [x]
            total_number_of_pages[x] = realloc(total_number_of_pages[x],
            total_number_of_books[x] * sizeof(int));
            total_number_of_pages[x][total_number_of_books[x] - 1] = y;
        } else if (type_of_query == 2) {
            int x, y;
            scanf("%d %d", &x, &y);
            printf("%d\n", (*(total_number_of_pages + x) + y));
        } else {
            int x;
            scanf("%d", &x);
            printf("%d\n", *(total_number_of_books + x));
        }
    }
}
```

---

---

**Roll Number:**

**Page: 94**

---

---

**Roll Number:**

**Page: 61**

```
}

if (total_number_of_books) {
    free(total_number_of_books);
}
for (int i = 0; i < total_number_of_shelves; i++) {
    if (*(total_number_of_pages + i)) {
        free(*(total_number_of_pages + i));
    }
}
if (total_number_of_pages) {
    free(total_number_of_pages);
}
return 0; }
```

**Output:**

```
5
5
1 0 15
1 0 20
1 2 78
2 2 0
3 0
```

```
78 2
```



## Practice – 11

### Structure, Union, typedef, bit-fields and enum

(a) Write a c program to find the total, average of n students using structures

```
#include <stdio.h>
// Define the structure for student information
struct Student {   char name[50];   int
marks;
}; int
main() {
int n;
    // Get the number of students
printf("Enter the number of students: ");
scanf("%d", &n);
    // Declare an array of structures to store student information
struct Student students[n];   // Input information for each
student
    for (int i = 0; i < n; i++) {
        printf("Enter the name of student %d: ", i + 1);
scanf("%s", students[i].name);
        printf("Enter the marks for %s: ", students[i].name);
scanf("%d", &students[i].marks);
    }
    // Calculate total and average marks
int total = 0;   for (int i =
0; i < n; i++) {
        total += students[i].marks;
    }
float average = (float)total / n;   //
Display the total and average marks
printf("\nTotal marks: %d\n", total);
printf("Average marks: %.2f\n", average);
return 0; }
```

Output:

```
Enter the number of students: 3
Enter the name of student 1: Alice
Enter the marks for Alice: 90
Enter the name of student 2: Bob
Enter the marks for Bob: 76
Enter the name of student 3: Cathy
Enter the marks for Cathy: 47
Total marks: 213
Average marks: 71.00
```

**(b) C program for Boxes through Tunnel**

Objective: Using a structure for transporting some boxes through a tunnel

<https://www.hackerrank.com/challenges/too-high-boxes/problem?isFullScreen=true>

```
#include <stdio.h> #define
maxHeight 41 struct Box
{
    int length, width, height;
};
int volume(struct Box box) {
    return box.length*box.width*box.height;
} int lower(struct Box box) {
    return box.height < maxHeight;
}

int main() {
    int n;
    printf("Enter number of boxes: ");
    scanf("%d", &n);    struct Box
boxes[100];
    printf("Enter %d boxes dimensions\n", n);
    for (int i = 0; i < n; i++) {
        printf("Enter length, width and height of Box - %d: ", i + 1);
        scanf("%d%d%d", &boxes[i].length, &boxes[i].width, &boxes[i].height);
    }
    printf("Areas of the Boxes that can passthrough the tunnel\n");
    for (int i = 0; i < n; i++) {        if (lower(boxes[i]))
        printf("%d\n", volume(boxes[i]));
    }
    return 0; }
```

**Output:**

```
Enter number of boxes: 4
Enter 4 boxes dimensions
Enter length, width and height of Box - 1: 5 5 5
Enter length, width and height of Box - 2: 1 2 40
Enter length, width and height of Box - 3: 10 5 41 Enter
length, width and height of Box - 4: 7 2 42
```

```
Areas of the Boxes that can passthrough the tunnel
125
80
```

**(c) C program for Post Transition**

Objective: Storing and transferring packages using pointers in structures.

<https://www.hackerrank.com/challenges/post-transition/problem?isFullScreen=true>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STRING_LENGTH 6

struct package
{   char* id;
    int weight;
};

typedef struct package package;

struct post_office
{   int min_weight;
    int max_weight;
    package* packages;
    int packages_count;
};

typedef struct post_office post_office;

struct town {   char*
name;   post_office*
offices;   int
offices_count;
};

typedef struct town town;

void print_all_packages(town t)
{
    printf("%s:\n", t.name);
    for (int i = 0; i < t.offices_count; i++)
    {
        printf("\t%d:\n", i);        for (int j = 0; j <
t.offices[i].packages_count; j++)
        printf("\t\t%s\n", t.offices[i].packages[j].id);    }
    }

void send_all_acceptable_packages(town* source, int source_office_index, town* target, int
target_office_index)
{   int n =
0;
    for (int i = 0; i < source->offices[source_office_index].packages_count; i++)
    if (source->offices[source_office_index].packages[i].weight >= target-
```

```

>offices[target_office_index].min_weight &&
    source->offices[source_office_index].packages[i].weight <=
target->offices[target_office_index].max_weight)
    ++n;
    package* newPackages = malloc(sizeof(package)*(n + target-
>offices[target_office_index].packages_count));
    package* oldPackages =
malloc(sizeof(package)*(source->offices[source_office_index].packages_count
- n));    for (int i = 0; i < target->offices[target_office_index].packages_count;
i++)    newPackages[i] = target->offices[target_office_index].packages[i];
n = target->offices[target_office_index].packages_count;    int m = 0;    for
(int i = 0; i < source->offices[source_office_index].packages_count; i++)
if (source->offices[source_office_index].packages[i].weight >= target-
>offices[target_office_index].min_weight &&
    source->offices[source_office_index].packages[i].weight <=
target->offices[target_office_index].max_weight)
    {
        newPackages[n] = source->offices[source_office_index].packages[i];
        ++n;
    }    else
    {
        oldPackages[m] = source->offices[source_office_index].packages[i];
        ++m;
    }

    target->offices[target_office_index].packages_count = n;
    free(target->offices[target_office_index].packages);    target-
>offices[target_office_index].packages = newPackages;    source-
>offices[source_office_index].packages_count = m;    free(source-
>offices[source_office_index].packages);    source-
>offices[source_office_index].packages = oldPackages; }

int number_of_packages(town t)
{
    int ans = 0;
    for (int i = 0; i < t.offices_count; i++)
        ans += t.offices[i].packages_count;
    return ans;
}

town town_with_most_packages(town* towns, int towns_count)
{
    int ans;    int max_packages = -1;
    for (int i = 0; i < towns_count; i++)
        if (number_of_packages(towns[i]) > max_packages)
            {

```

```

        max_packages = number_of_packages(towns[i]);
    ans = i;
    }
    return towns[ans];
}

town* find_town(town* towns, int towns_count, char* name)
{
    for (int i = 0; i < towns_count;
    i++)
        if (!strcmp(towns[i].name,
        name))
            return &(towns[i]);
    return &towns[0];
}

int main() {
    int towns_count;
    scanf("%d", &towns_count);
    town* towns = malloc(sizeof(town)*towns_count);
    for (int i = 0; i < towns_count; i++) {
        towns[i].name = malloc(sizeof(char) * MAX_STRING_LENGTH);
        scanf("%s", towns[i].name);
        scanf("%d", &towns[i].offices_count);
        towns[i].offices = malloc(sizeof(post_office)*towns[i].offices_count);
        for (int j = 0; j < towns[i].offices_count; j++) {
            scanf("%d%d%d", &towns[i].offices[j].packages_count,
            &towns[i].offices[j].min_weight, &towns[i].offices[j].max_weight);
            towns[i].offices[j].packages =
            malloc(sizeof(package)*towns[i].offices[j].packages_count);
            for (int k = 0; k < towns[i].offices[j].packages_count; k++) {
                towns[i].offices[j].packages[k].id = malloc(sizeof(char) *
                MAX_STRING_LENGTH);
                scanf("%s", towns[i].offices[j].packages[k].id);
                scanf("%d", &towns[i].offices[j].packages[k].weight);
            }
        }
        int queries;
        scanf("%d", &queries);
        char town_name[MAX_STRING_LENGTH];
        while (queries--) {
            int type;
            scanf("%d", &type);
            switch (type) {
                case
                1:
                    scanf("%s", town_name);
                    town* t = find_town(towns, towns_count, town_name);
                    print_all_packages(*t);
                    break;
                case 2:

```

```

scanf("%s", town_name);
town* source = find_town(towns, towns_count, town_name);
int source_index;    scanf("%d", &source_index);
scanf("%s", town_name);
town* target = find_town(towns, towns_count, town_name);
int target_index;    scanf("%d", &target_index);
send_all_acceptable_packages(source, source_index, target, target_index);
break;    case 3:
    printf("Town with the most number of packages is %s\n",
town_with_most_packages(towns, towns_count).name);
break;
    } }
return 0; }

```

**(d) copy one structure variable to another of same type**

```

#include <stdio.h>
struct class {    int
no;    char name
[20];    float
marks;
};

int main () {    int x;    struct class stu1 = {111, "Rao", 72.50};
struct class stu2 = {222, "Reddy", 67.80};    struct class stu3;
stu3 = stu2;    x = ((stu3.no == stu2.no) && (stu3.marks ==
stu2.marks))? 1 : 0;    if (x==1)
{
    printf("student 2 & student 3 are same \n");    printf
("%d\t%s\t%f ", stu3.no, stu3.name, stu3.marks);
    }    else    printf ("student 2 and student 3 are
different "); }

```

**Output:**

Student 2 and student 3 are same  
222 Reddy 67.8000

**(e) Read student name and marks from the command line and display the student details, along with the total.**

```

#include <stdio.h> typedef
struct Stu {

```

```
        char
name[20];    int age;
int marks[5];
        int total;
} Student;

int main()
{
    Student s1;
    printf("Enter student name: ");
scanf("%s", s1.name);    printf("Enter
student age: ");
    scanf("%d", &s1.age);
    printf("Enter student marks in 5 subjects: ");
    int i;
    for (i = 0; i < 5; i++)
    {
        scanf("%d", &s1.marks[i]);
    }
    // calculating total
    for (i = 0; i < 5; i++)
    {
        s1.total += s1.marks[i];
    }
    printf("Student Details\nName\tAge\tTotal\n-----\n");
printf("%s\t%d\t%d", s1.name, s1.age, s1.total); }
```

**Output:**

```
Enter student name: Bob
Enter student age: 17
Enter student marks in 5 subjects: 54 37 64 92 41
Student Details
Name  Age  Total
-----
Bob   17   288
```

---

---

**Roll Number:**

**Page: 70**

**(f) Shift/rotate using bitfields.**

```
#include <stdio.h> union
test {
    // Bit-field member x with 3 bits
    unsigned int x : 3;
    // Bit-field member y with 3 bits
    unsigned int y : 3;
    // Regular integer member z
    int z;
};

int main()
{
    // Declare a variable t of type union test
    union test t;
    // Assign the value 5 to x (3 bits)
    t.x = 5;
    // Assign the value 4 to y (3 bits)
    t.y = 4;
    // Assign the value 1 to z (32 bits)
    t.z = 1;
    // Print the values of x, y, and z
    printf("t.x = %d, t.y = %d, t.z = %d", t.x, t.y, t.z);
    return 0;
}
```

**Output**

t.x = 1, t.y = 1, t.z = 1



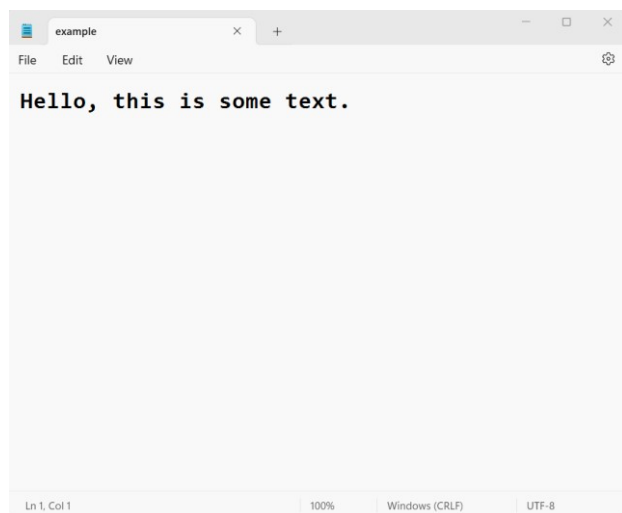
## Practice - 12

## Files

## (a) Write text into and read text from a file

```
#include <stdio.h>
int main() {
    FILE *file;
    char textToWrite[] = "Hello, this is some text.";
    char buffer[100]; // Writing text to a file
    file = fopen("example.txt", "w");
    if (file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }
    fprintf(file, "%s", textToWrite);
    fclose(file); // Reading text from a file
    file = fopen("example.txt", "r");
    if (file == NULL) {
        perror("Error opening file for reading");
        return 2;
    }
    fgets(buffer, sizeof(buffer), file);
    fclose(file); // Print the read text
    printf("Read from file: %s\n", buffer);
    return 0; }
```

## example.txt file



## Output:

Read from file: Hello, this is some text.

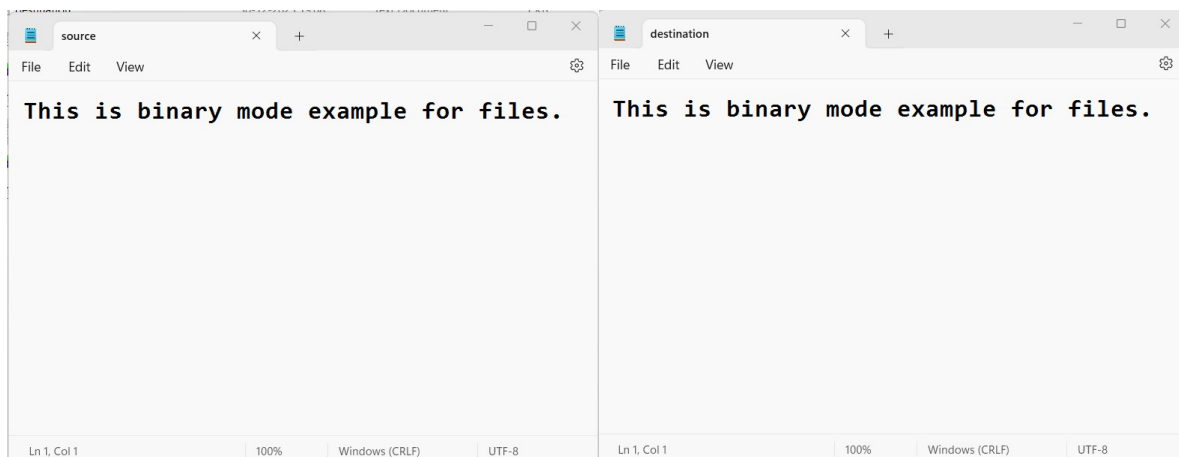
## (b) Write into text and read text from binary file using fread() and fwrite()

```
#include <stdio.h>
int main() {
```

```

FILE *sourceFile, *destinationFile;  char
buffer[1024]; // Buffer to store data  // Open the
source file for reading in binary mode  sourceFile
= fopen("source.txt", "rb");  if (sourceFile ==
NULL) {      perror("Error opening source file");
return 1;
}
// Open the destination file for writing in binary mode
destinationFile = fopen("destination.txt", "wb");  if
(destinationFile == NULL) {      perror("Error
opening destination file");      fclose(sourceFile);
return 2;
}
// Copy the contents of the source file to the destination file
size_t bytesRead;
while ((bytesRead = fread(buffer, 1, sizeof(buffer), sourceFile)) > 0) {
fwrite(buffer, 1, bytesRead, destinationFile);
}
fclose(sourceFile);
fclose(destinationFile);
return 0; }

```

**Output:****source.txt file****destination.txt file****(c) Copy the contents of one file to another file**

```

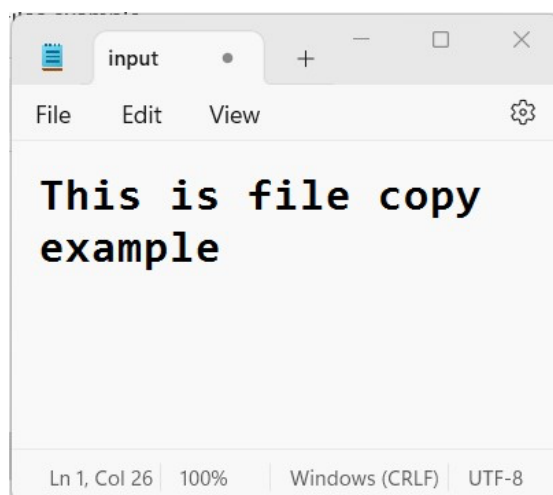
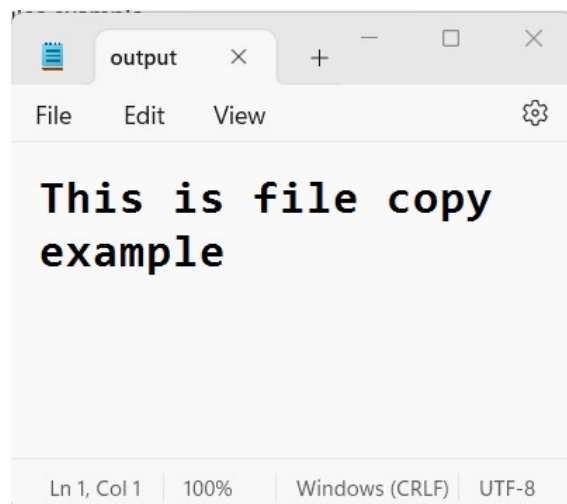
#include <stdio.h> int
main() {
    FILE *sourceFile, *destinationFile;
    char ch;
    // Open the source file for reading
    sourceFile = fopen("input.txt", "r");  if
    (sourceFile == NULL) {
        perror("Error opening source file");
        return 1;
    }
}

```

```

// Open the destination file for writing
destinationFile = fopen("output.txt", "w");
if (destinationFile == NULL) {
    perror("Error opening destination file");
    fclose(sourceFile);
    return 2;
}
// Copy contents from source to destination
while ((ch = fgetc(sourceFile)) != EOF) {
    fputc(ch, destinationFile);
}
// Close the files
fclose(sourceFile);
fclose(destinationFile); printf("File
copied successfully.\n"); return 0;
}

```

**Output:****input.txt file****output.txt file****(d) Merge two files into the third file using command line arguments**

```

#include <stdio.h> int main(int
argc, char *argv[]) {
    if (argc != 4) {
        printf("Usage: %s source1.txt source2.txt destination.txt\n", argv[0]);
        return 1;
    }
    FILE *sourceFile1, *sourceFile2, *destinationFile;
    char ch;
    // Open the first source file for reading
    sourceFile1 = fopen(argv[1], "r"); if
    (sourceFile1 == NULL) { perror("Error
    opening first source file"); return 2;
    }
}

```

```

    // Open the second source file for
    reading    sourceFile2 = fopen(argv[2],
    "r");    if (sourceFile2 == NULL) {
        perror("Error opening second source file");
    fclose(sourceFile1);
        return 3;
    }
    // Open the destination file for writing
    destinationFile = fopen(argv[3], "w");    if
    (destinationFile == NULL) {
    perror("Error opening destination file");
    fclose(sourceFile1);
    fclose(sourceFile2);
        return 4;
    }
}

```

### Practice - 13

#### Augmented Experiments

##### (a) Variadic functions in C

Objective: Understanding variable number of arguments

<https://www.hackerrank.com/challenges/variadic-functions-inc/problem?isFullScreen=true>

```

#include <stdio.h>
#include <stdarg.h>
// Function to find the average of a variable number of integers double
findAverage(int num, ...) {
    va_list args;    va_start(args,
    num);    double sum = 0;
    for (int i = 0; i < num; i++) {
        sum += va_arg(args, int);
    }
    va_end(args);
    return sum / num;
} int
main() {
    // Example usage of the variadic function
    double average1 = findAverage(3, 10, 20, 30);
    double average2 = findAverage(2, 10, 20);
    double average3 = findAverage(6, 100, 200, 300, 400, 500, 600);
    printf("Average: %.2f\n", average1);    printf("Average: %.2f\n",
    average2);    printf("Average: %.2f", average3);
    return 0; }

```

**Output:**

Average: 20.00  
 Average: 15.00  
 Average: 350.00

### (b) Small triangles, Large Triangles

Objective: Print sorted by their areas

<https://www.hackerrank.com/challenges/small-triangles-large-triangles/problem?isFullScreen=true>

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

struct triangle {
    int a;    int b;    int c; };
typedef struct triangle
triangle;
void sort_by_area(triangle* tr, int n) {
    /**
     * Sort an array a of the length n
     */
    float areas[n];
    for (int i = 0; i < n; i++) {
        float p = (tr[i].a + tr[i].b + tr[i].c) / 2.0;
        areas[i] = sqrt(p * (p - tr[i].a) * (p - tr[i].b) * (p - tr[i].c));
    }
    // Applying bubble sort
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (areas[j] > areas[j + 1]) {
                // swapping area
                float t = areas[j];
                areas[j] = areas[j + 1];
                areas[j + 1] = t;

                // swapping triangles
                triangle temp = tr[j];
                tr[j] = tr[j + 1];
                tr[j + 1] = temp;
            }
        }
    }
}

int main()
{
    int n, i;
```

```
printf("Enter number of triangles: ");
scanf("%d", &n);
triangle *tr = malloc(n * sizeof(triangle)); for
(i = 0; i < n; i++) { printf("Enter sides of
triangle %d: ", i + 1); scanf("%d%d%d",
&tr[i].a, &tr[i].b, &tr[i].c);
}
sort_by_area(tr, n);
printf("After Sorting the given triangles based on areas: \n");
for (i = 0; i < n; i++) {
printf("%d %d %d\n", tr[i].a, tr[i].b, tr[i].c);
}
return 0; }
```

**Output:**

```
Enter number of triangles: 3
Enter sides of triangle 1: 7 24 25
Enter sides of triangle 2: 5 12 13
Enter sides of triangle 3: 3 4 5
After Sorting the given triangles based on areas:
3 4 5
5 12 13
7 24 25
```