**WEEK1**

**Q1.Jack and John received a collection of gifts from their friends after a party. Each**

**gift has a unique numeric value associated with it. However, Alice and Bob**

**noticed that there is one gift value that occurs an odd number of times, while all**

**other gift values occur an even number of times. Write a program to help Alice**

**and Bob find out the gift value that occurs an odd number of times in optimal**

**Time.**

**[ Take the input as: 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2].**

**CODE:**

#include <stdio.h>

int main() {

    int a[100];

    int n;

    printf("enter the number of gift values you want to enter: ");

    scanf("%d",&n);

    printf("enter the gift values");

    for(int i=0;i<n;i++){

        scanf("%d",&a[i]);

    }

    int b[100];

    for(int i=0;i<100;i++){

        b[i]=0;

    }

    for (int i=0;i<n;i++){

        b[a[i]]++;

    }

    for(int i=0;i<100;i++){

        if (b[i]%2!=0)

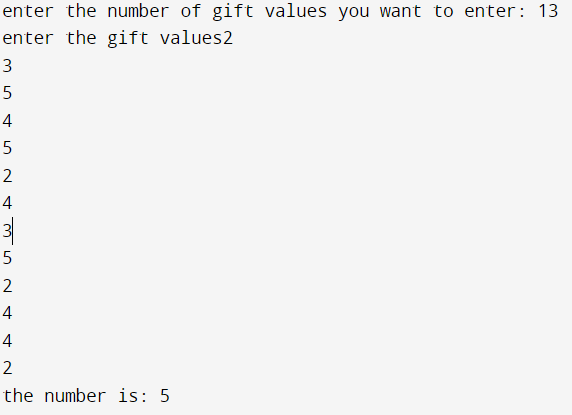
            printf("the number is: %d",i);

    }

    return 0;

}

**OUTPUT:**



**Complexity: O(n)**

**Q2. In a sports event, 100 players participated and scored some points. Each player scored 5 points more than his predecessor. The first player got 5 points. Write a program to find the total points scored by all the players in optimal time.**

**CODE:**

#include <stdio.h>

int main() {

    int a[100];

    a[0]=5;

    int total;

    for(int i=1;i<100;i++){

        a[i]=a[i-1]+5;

    }

    for (int i=0;i<100;i++){

        total+=a[i];

    }

    printf("the total is:%d",total);

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/nf2piMuq8sW3YErE0ovwyXDPYRYnoOrchnDAuiE_wjiKLlPVo779x08XK1tTTNw_FXwHuJKoOg7I5LtlMeJrtfQqg3EI3IAUXGlPPU0yUVvqZlYhZRuOBGRuhyVxetOhvFG98gPEFkdi8FCQQHnwCxkDE6L2AZVp

**Complexity: O(n)**

**Q3.** **You are given a list of student names along with their corresponding grades. Your task is to implement a function find\_student\_grade that takes in a list of student names and a target student name. The function should return the grade of the target student. If the target student is not found in the list, the function should return “Not Found”.**

**CODE:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct student{

    char \*name;

    int grade;

}S;

S \*find\_student\_grade(S \*students, int num, char \*target\_student){

    for(int i=0;i<num;i++){

        if(strcmp(students[i].name,target\_student)==0){

            return &students[i];

        }

    }

    return NULL;

}

int main() {

     S students[] = {

        {"Alice", 90},

        {"Bob", 80},

        {"Carol", 70},

        {"Dave", 60},

        {"Eve", 50},

        };

    int num = sizeof(students) / sizeof(students[0]);

    char \*target\_student = "Bob";

    S \*student = find\_student\_grade(students, num, target\_student);

    if (student != NULL) {

        printf("%s's grade is %d\n", student->name, student->grade);

    } else {

        printf("Student not found\n");

  }

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/eCB3RKTYEnOQsM5WOxrIKKEI0rVUW6UHiKp-3lGnv8Qs8nDGMLnOUq9J_wDx8vGWpnmpNoh666sQ38vdI88tyFmdsuC2ozWUrgoZQRqYZ4wYFVTO3x6t9AmjQu5j3ro-pSkQpaH0WUl6FB3JN1t0FxJVz4XGHDor

**Q4.**

**Write a program using iterative approach to calculate and display the total rewards Alice and Bob will earn based on the number of items they purchase. Discuss if the solution for the**

**problem can be optimized in terms of time complexity. If yes, then write a program for the same with optimized approach.**

**CODE:**

int main() {

    int m,n;

    int ta=0;

    int tb=0;

    printf("enter the number of items in Alice's list");

    scanf("%d",&n);

    printf("enter the number of items in Bob's list");

    scanf("%d",&m);

    for(int i=1;i<=n;i++){

        ta+=(4\*i);

    }

    for(int i=1;i<=m;i++){

        tb+=4\*i;

    }

    printf("total rewards earned by Alice : %d \n",ta);

    printf("total rewards earned by Bob : %d \n",tb);

    //approach that takes less time complexity

    int sa,sb;

    sa=(n\*(4+(4\*n)))/2;

    sb=(m\*(4+(4\*m)))/2;

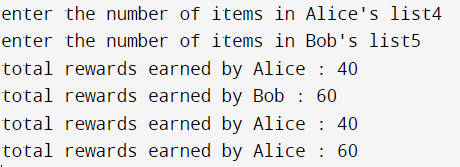
    printf("total rewards earned by Alice : %d \n",sa);

    printf("total rewards earned by Alice : %d \n",sb);

    return 0;

}

**OUTPUT:**



(Here first sum has time complexity O(n) due to iterative method. Second attempt uses arithmetic progression hence has time complexity O(1).)

**WEEK 2**

**Q1. The array data structure can be used to store multiple values. Data can be searched or modified.**

**A specific data part can be deleted, New Data can always be inserted. However, all these operations**

**may not be efficient. In this lab, you need to figure out the number of operations (steps of data**

**movement) needed for various types of jobs. In a later lab, we shall see the difference when a**

**different efficient way would be used.**

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

struct emp\_info{

    char name[100];

    int id;

    char department[100];

    int salary;

    char address[100];

    int contact;

    char email[100];

    char position[100];

    int exp;

}emp[50];

int count;

void read\_file(){

    FILE \*fp=fopen(&quot;lab2.txt&quot;,&quot;r&quot;);

    if(fp == NULL){

        printf(&quot;FILE EMPTY&quot;);

    }

    else{

        char c;

        count = 0;

        for (c = getc(fp);c!=EOF;c = getc(fp)){

            if(c == &#39;\n&#39;){

                count = count + 1;

            }

        }

        printf(&quot;%d\n&quot;,count);

        fseek(fp,0,SEEK\_SET);

        for(int i = 0;i&lt;count;i++){

            fscanf(fp, &quot;%s %d %s %d %s %d %s %s

%d&quot;,emp[i].name,&amp;emp[i].id,emp[i].department,&amp;emp[i].salary,emp[i].address,&amp;em

p[i].contact,emp[i].email,emp[i].position,&amp;emp[i].exp);

        }

        fclose(fp);

    }

}

void first\_emp(){

    printf(&quot;\nFIRST EMPLOYEE\n%s %d %s %d %s %d %s %s

%d\n&quot;,emp[0].name,emp[0].id,emp[0].department,emp[0].salary,emp[0].address,emp

[0].contact,emp[0].email,emp[0].position,emp[0].exp);

}

void last\_emp(){

    int x = count-1;

    printf(&quot;\nLAST EMPLOYEE\n%s %d %s %d %s %d %s %s

%d\n&quot;,emp[x].name,emp[x].id,emp[x].department,emp[x].salary,emp[x].address,emp

[x].contact,emp[x].email,emp[x].position,emp[x].exp);

}

void del\_last(){

    count = count-1;

    printf(&quot;\nsucessfully deleted last&quot;);

    printf(&quot;\nNow total number of data in structure are %d\n&quot;,count);

    printf(&quot;Records of last employee are ignored/deleted\n&quot;);

}

void del\_first(){

    int c = 0;

    for(int i = 0; i&lt;count-1; i++){

        strcpy(emp[i].name,emp[i+1].name);

        emp[i].id = emp[i+1].id;

        strcpy(emp[i].department,emp[i+1].department);

        emp[i].salary = emp[i+1].salary;

        strcpy(emp[i].address,emp[i+1].address);

        emp[i].contact = emp[i+1].contact;

        strcpy(emp[i].email,emp[i+1].email);

        strcpy(emp[i].position,emp[i+1].position);

        emp[i].exp = emp[i+1].exp;

        c++;

    }

    count = count-1;

    printf(&quot;\ntotal records after deletion of first: %d&quot;,count);

    printf(&quot;\nloop counts: %d&quot;,c);

    printf(&quot;\ntotal data transfer counts: %d\n&quot;,c\*9);

}

void del\_third(){

    int c=0;

    for(int i = 2; i&lt;count-1; i++){

        strcpy(emp[i].name,emp[i+1].name);

        emp[i].id = emp[i+1].id;

        strcpy(emp[i].department,emp[i+1].department);

        emp[i].salary = emp[i+1].salary;

        strcpy(emp[i].address,emp[i+1].address);

        emp[i].contact = emp[i+1].contact;

        strcpy(emp[i].email,emp[i+1].email);

        strcpy(emp[i].position,emp[i+1].position);

        emp[i].exp = emp[i+1].exp;

        c++;

    }

    count = count-1;

    printf(&quot;\nthird record deleted&quot;);

    printf(&quot;\ntotal loop condition checks: %d&quot;,c+1);

    printf(&quot;\ntotal number of data transfers: %d&quot;,c\*9);

    printf(&quot;\ntotal records after deletion: %d\n&quot;,count);

}

void insert\_last(){

    if (count+1 &lt;= 50){

        printf(&quot;\nINSERT AT LAST\n&quot;);

        printf(&quot;enter employee name &quot;);

        scanf(&quot;%s&quot;,emp[count].name);

        printf(&quot;enter employee id &quot;);

        scanf(&quot;%d&quot;,&amp;emp[count].id);

        printf(&quot;enter employee department &quot;);

        scanf(&quot;%s&quot;,emp[count].department);

        printf(&quot;enter employee salary &quot;);

        scanf(&quot;%d&quot;,&amp;emp[count].salary);

        printf(&quot;enter employee address (no spaces) &quot;);

        scanf(&quot;%s&quot;,emp[count].address);

        printf(&quot;enter employee contact &quot;);

        scanf(&quot;%d&quot;,&amp;emp[count].contact);

        printf(&quot;enter employee email &quot;);

        scanf(&quot;%s&quot;,emp[count].email);

        printf(&quot;enter employee position &quot;);

        scanf(&quot;%s&quot;,emp[count].position);

        printf(&quot;enter employee work experience &quot;);

        scanf(&quot;%d&quot;,&amp;emp[count].exp);

        count = count+1;

        printf(&quot;\nnumber of data additions in inserting at last: 9\n&quot;);

        printf(&quot;successfully inserted at last\n&quot;);

    }

    else{

        printf(&quot;cannot enter more records&quot;);

    }

}

void insert\_first(){

    int c= 0;

    for(int i = count-1; i&gt;=0; i--){

        strcpy(emp[i+1].name,emp[i].name);

        emp[i+1].id = emp[i].id;

        strcpy(emp[i+1].department,emp[i].department);

        emp[i+1].salary = emp[i].salary;

        strcpy(emp[i+1].address,emp[i].address);

        emp[i+1].contact = emp[i].contact;

        strcpy(emp[i+1].email,emp[i].email);

        strcpy(emp[i+1].position,emp[i].position);

        emp[i+1].exp = emp[i].exp;

        c++;

    }

    printf(&quot;\nINSERT AT FIRST\n&quot;);

    printf(&quot;enter employee name &quot;);

    scanf(&quot;%s&quot;,emp[0].name);

    printf(&quot;enter employee id &quot;);

    scanf(&quot;%d&quot;,&amp;emp[0].id);

    printf(&quot;enter employee department &quot;);

    scanf(&quot;%s&quot;,emp[0].department);

    printf(&quot;enter employee salary &quot;);

    scanf(&quot;%d&quot;,&amp;emp[0].salary);

    printf(&quot;enter employee address (no spaces) &quot;);

    scanf(&quot;%s&quot;,emp[0].address);

    printf(&quot;enter employee contact &quot;);

    scanf(&quot;%d&quot;,&amp;emp[0].contact);

    printf(&quot;enter employee email &quot;);

    scanf(&quot;%s&quot;,emp[0].email);

    printf(&quot;enter employee position &quot;);

    scanf(&quot;%s&quot;,emp[0].position);

    printf(&quot;enter employee work experience &quot;);

    scanf(&quot;%d&quot;,&amp;emp[0].exp);

    count = count +1;

    printf(&quot;\nsuccessfully inserted at first&quot;);

    printf(&quot;\nloop counts: %d&quot;,c);

    printf(&quot;\ntotal data transfer counts (including data insertion/addition):

%d&quot;,(c\*9)+9);

    printf(&quot;\ntotal employees: %d\n&quot;,count);

}

void insert\_second(){

    int c = 0;

    for(int i = 1; i&lt;count-1; i++){

        strcpy(emp[i+1].name,emp[i].name);

        emp[i+1].id = emp[i].id;

        strcpy(emp[i+1].department,emp[i].department);

        emp[i+1].salary = emp[i].salary;

        strcpy(emp[i+1].address,emp[i].address);

        emp[i+1].contact = emp[i].contact;

        strcpy(emp[i+1].email,emp[i].email);

        strcpy(emp[i+1].position,emp[i].position);

        emp[i+1].exp = emp[i].exp;

        c++;

    }

    printf(&quot;\nINSERT AT SECOND\n&quot;);

    printf(&quot;enter employee name &quot;);

        scanf(&quot;%s&quot;,emp[1].name);

        printf(&quot;enter employee id &quot;);

        scanf(&quot;%d&quot;,&amp;emp[1].id);

        printf(&quot;enter employee department &quot;);

        scanf(&quot;%s&quot;,emp[1].department);

        printf(&quot;enter employee salary &quot;);

        scanf(&quot;%d&quot;,&amp;emp[1].salary);

        printf(&quot;enter employee address (no spaces) &quot;);

        scanf(&quot;%s&quot;,emp[1].address);

        printf(&quot;enter employee contact &quot;);

        scanf(&quot;%d&quot;,&amp;emp[1].contact);

        printf(&quot;enter employee email &quot;);

        scanf(&quot;%s&quot;,emp[1].email);

        printf(&quot;enter employee position &quot;);

        scanf(&quot;%s&quot;,emp[1].position);

        printf(&quot;enter employee work experience &quot;);

        scanf(&quot;%d&quot;,&amp;emp[1].exp);

    count = count +1;

    printf(&quot;\nloop counts: %d&quot;,c);

    printf(&quot;\ntotal data transfer counts (includeing insertion/addition):

%d\n&quot;,(c\*9)+9);

}

void search\_four(){

    int arr[count];

    for(int i = 0; i&lt;count; i++){

        arr[i]=emp[i].id;

    }

    for (int i = 0; i &lt; count; ++i){

        for (int j = i + 1; j &lt; count; ++j){

            if (arr[i] &gt; arr[j]){

                int a =  arr[i];

                arr[i] = arr[j];

                arr[j] = a;

            }

        }

    }

    int ele = arr[3];

    int coun = 0;

    for(int i = 0;i&lt;count;i++){

        if(emp[i].id == ele){

            printf(&quot;\nemployee name with id %d: %s\n&quot;,emp[i].id,emp[i].name);

            break;

        }

        coun++;

    }

    printf(&quot;loop condition checks: %d\n&quot;,coun+1);

}

int main(){

    read\_file();

    first\_emp();

    last\_emp();

    search\_four();

    del\_first();

    first\_emp();

    last\_emp();

    insert\_first();

    first\_emp();

    last\_emp();

    del\_last();

    first\_emp();

    last\_emp();

    insert\_last();

    first\_emp();

    last\_emp();

    del\_third();

    insert\_second();

    first\_emp();

    last\_emp();

    return(0);

}

**lab2.txt**

n1 1 d1 807 address1 9548228 a1@mail pos1 1

n2 2 d2 1000 address2 9876546 a2@mail pos2 2

n3 3 d3 9461 addrss3 6310298 a3@mail pos3 3

n4 5 d4 21323 address4 98298773 a4@mail pos4 4

n5 4 d5 9823 address5 9749831 h5@mail pos5 5

n6 6 d6 31231 address6 212321 a6@mail pos6 6

n7 7 d7 94123 address7 1267893 a7@mail pos7 7

n8 8 d8 565665 address8 333333 g8@mail pos8 8

n9 9 d9 121254 address9 222222 a9@mail pos9 9

n10 10 d10 34434 address10 444444 a10@mail pos10 10

n11 11 d11 321332 address11 5555555 a11@mail pos11 11

n12 12 d12 889787 address12 6666666 u12@mail pos12 12

n13 13 d13 873927 address13 777777 a13@mail pos13 13

n14 14 d14 767676 address14 888888 d14@mail pos14 14

n15 15 d15 676767 address15 999999 n15@mail pos15 15

n16 16 d16 12122 address16 0011111 s16@mail pos16 16

n17 17 d17 224928 address17 012133 r17@mail pos17 17

n18 18 d18 112132 address18 939283 n18@mail pos18 18

n19 19 d19 213131 address19 9928932 a19@mail pos19 19

n20 20 d20 982 address20 3283299 a20@mail pos20 20

n21 21 d21 8323 address21 88232 n21@mail pos21 21

n22 22 d22 9232 address22 8271311 e22@mail pos22 22

n23 23 d23 92131 address23 992883 f23@mail pos23 23

n24 24 d24 123123 address24 1123912 n24@mail pos24 24

n25 25 d25 11111 address25 982983 n25@mail pos25 25

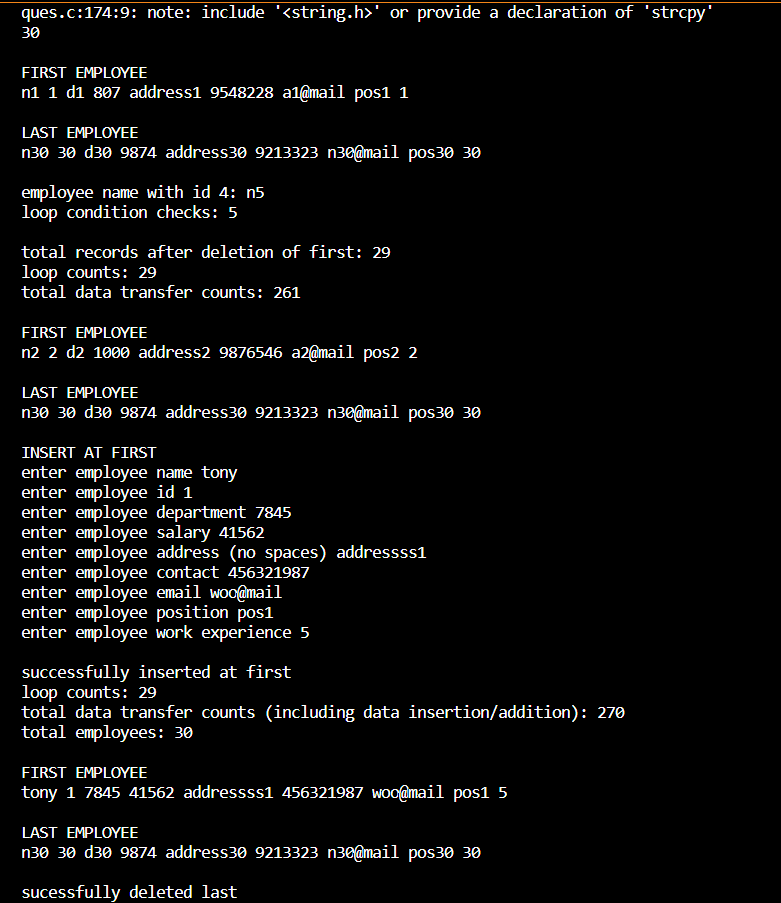
n26 26 d26 19238 address26 998923 n26@mail pos26 26

n27 27 d27 66662 address27 291383 v27@mail pos27 27

n28 28 d28 21321 address28 82388783 c28@mail pos28 28

n29 29 d29 72391 address29 231322 n29@mail pos29 29

n30 30 d30 9874 address30 9213323 n30@mail pos30 30

**OUTPUT:**

**WEEK 3**

**Q1.In a sport event, 5 teams are participating and each team is denoted by a number i.e., team 1 is denoted by ‘1’, team 2 is denoted by ‘2’ and likewise for rest of the teams. Whenever any team scores a point, then its assigned number is added into a list. The most occurring team number in the list will decide the winner of the match.**

**Final list after the completion of the match is given below:**

**[5, 6, 3, 4, 2, 3, 4, 5, 1, 3]**

**Write a program to find the winner of the match.**

**Output: 3.**

**CODE-**

#include <stdio.h>

int main() {

    int a[]={5, 6, 3, 4, 2, 3, 4, 5, 1, 3};

    int max;

    int count=0;

    int b[10];

    for (int i=0;i<10;i++){

        b[i]=0;

    }

    for (int i=0;i<10;i++){

        b[a[i]]++;

    }

    for (int i=0;i<10;i++){

        if(b[i]>max)

            max=b[i];

    }

    printf("the winner is: %d",max);

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/vaW1WEZExzVHTLRuwKuk8hM21m5NqJI1UkkEyQwaBXslZgG6uK9dWxA_1T1qXU0RCohXO_7dKFhURPrIJ49E03h8hWtAmhBg01Co9T-BMfCuI-2FnUYlex5S_vofRDT3Ymy1-ceMQVjbPpitlDgs_vvsKQtrBRt8

**Q2.** **In a class, marks obtained by 8 students in an exam is listed below:**

**[70, 40,67,38,89,56,78,45]**

**Write a program to find the scores of students which are having at least two student’s score greater than it.**

**CODE:**

#include <stdio.h>

int main() {

    int a[]={70,40,67,38,89,56,78,45};

    int temp;

    for (int i=0;i<8;i++){

        for(int j=0;j<8-i-1;j++)

            if (a[j]>a[j+1]){

                temp=a[j];

                a[j]=a[j+1];

                a[j+1]=temp;

            }

        }

    for(int i = 0; i<8/2; i++){

        temp = a[i];

        a[i] = a[8-i-1];

        a[8-i-1] = temp;

    }

    for (int i=0;i<8;i++){

        printf(" %d",a[i]);

    }

    printf("\n");

    int max=a[0];

    int secondmax=0;

    for (int i=0;i<8;i++){

        if (a[i]<max){

            secondmax=a[i];

            break;

        }

    }

    for (int i=0;i<8;i++){

        if(a[i]==max|| a[i]==secondmax){

            continue;

        }

        else{

            printf("%d ",a[i]);

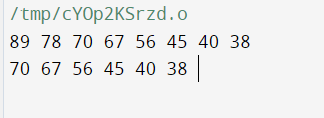
        }

    }

    return 0;

}

**OUTPUT:**



**Q3.** **The adventurers, Lily, Max, and Mia, were tasked with traversing through the Linked Lands**

**and documenting the population density each land. At final, they need to sum up the**

**population density of all the lands traversed. Write a program with a function named**

**calculate\_sum that takes the head of a linked list as its parameter. Each node contains an**

**integer value, and the next pointer points to the next node in the list. The program should**

**return the sum of all the integer values in the linked list.**

**Suppose they got the below data for each of the land.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node{

    int data;

    struct node \*next;

};

int insert(struct node \*head){

    struct node \*newnode,\*ptr;

    newnode=(struct node\*)malloc(sizeof(struct node));

    ptr=head;

    int x;

    printf("enter the population density: ");

    scanf("%d",&x);

    newnode->data=x;

    newnode->next=NULL;

    if(head==NULL){

        head=newnode;

    }

    else{

        while (ptr->next!=NULL){

            ptr=ptr->next;

        }

        newnode->data=x;

        ptr->next=newnode;

    }

    return head;

}

int sum=0;

void print(struct node \*head){

    if (head==NULL){

        printf("empty list");

    }

    else{

        struct node \*ptr=head;

        while(ptr!=NULL){

            sum+=ptr->data;

            ptr=ptr->next;

        }

        printf("the sum is: %d",sum);

    }

}

int main() {

    struct node \*head=NULL;

    head=insert(head);

    head=insert(head);

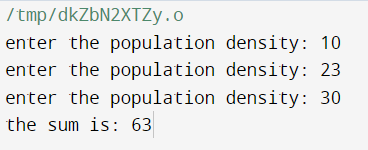
    head=insert(head);

    print(head);

    return 0;

}

**OUTPUT:**



**WEEK 4**

**Q1. In a class, all students are sitting according to their enrollment number in**

**ascending order staring from 1. Further for a group discussion session, they need**

**to be divided into two groups based on their even and odd position.**

**Write a program implementing Linked list to pull the list of all students with odd**

**positions followed by even positions.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node \*head = NULL;

struct node{

    int data;

    struct node \*next;

};

void insert(int ele){

    struct node \*temp = malloc(sizeof(struct node));

    temp -> data  = ele;

    temp -> next = NULL;

    if (head == NULL){

        head = temp;

    }

    else {

        struct node \*ptr = head;

        while(ptr -> next != NULL){

            ptr = ptr -> next;

        }

        ptr -> next = temp;

    }

}

void traverse(){

    if(head==NULL){

        printf("empty linked list");

    }

    else{

        struct node \*ptr = head;

        while(ptr != NULL){

            if(ptr->data % 2 == 1){

                printf("%d ",ptr->data);

                ptr = ptr->next;

                continue;

            }

            else{

                ptr = ptr->next;

            }

        }

        ptr = head;

        while(ptr != NULL){

            if(ptr->data % 2 == 0){

                printf("%d ",ptr->data);

                ptr = ptr->next;

                continue;

            }

            else{

                ptr = ptr->next;

            }

        }

    }

}

int main(){

    struct node \*head = NULL;

    insert(1);

    insert(2);

    insert(3);

    insert(4);

    insert(5);

    insert(6);

    traverse();

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/bY2ERJbFy6viXM9rIen9Pj10UYnlT_k14ZWTrBvasnD9FmVgOrSHc0tMeQ2sexawH6_FRGwBimhxPwbblY9rR8MAh1au6kHuvgliYo55XlYPugZPFhFmWC5Sq7QV4ce9NsiutybanvD_DfLlQr_Mok_czDHqQ5g9

**Q2.** **Josef and Joy are partying together. While playing the music, some of the songs**

**are present multiple time in the playlist. Order of the songs in the playlist is as**

**follow:**

**15 > > 12 > 15 > 12 > 43 > 21**

**Write a program implementing linked list to help Josef and Joy to remove the**

**duplicate songs from the playlist.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node \*next;

};

struct node \*head = NULL;

void insert(int ele){

    struct node \*temp = malloc(sizeof(struct node));

    temp -> data  = ele;

    temp -> next = NULL;

    if (head == NULL){

        head = temp;

    }

    else {

        struct node \*ptr = head;

        while(ptr -> next != NULL){

            ptr = ptr -> next;

        }

        ptr -> next = temp;

    }

}

void removeDuplicates(){

    struct node \*current=head, \*prev, \*duplicate;

    while (current != NULL && current->next != NULL) {

        prev = current;

        while (prev->next != NULL) {

            if (current->data == prev->next->data) {

                duplicate = prev->next;

                prev->next = prev->next->next;

            }

            else prev = prev->next;

        }

        current = current->next;

    }

}

void print\_list() {

    struct node \*current = head;

    while (current != NULL) {

        printf("%d ", current->data);

        current = current->next;

    }

    printf("\n");

}

int main() {

    insert(15);

    insert(16);

    insert(12);

    insert(15);

    insert(12);

    insert(43);

    insert(21);

    printf("Original list: ");

    print\_list();

    removeDuplicates();

    printf("List after removing duplicates: ");

    print\_list();

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/N-y6XbEG6oFGLnCZtPBxlhyTF7duMOuKHXFbaZ3OJt7g9VBjFGTwXxwfHpFvXj__SOdhDxAFO_TiP3s5pCuFuzjdrp4ks_s13fUB-NfukcBaQ21vtq-b_b6d_kDOaPCT5LLuy9QmKxKMwdhYTE-MfUTtaZ5xuJDf

**Q3.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int max(int num1, int num2){

    return (num1>num2)?num1:num2;

}

// Function to find the minimum number of jumps to reach the end of the pillar

int minJumps(int pillar[], int n)

{

    // If the monkey is at the first pillar, it can jump to any pillar

    // up to the maximum jump distance

    int maxJump = pillar[0];

    int jumps = 1;

    // Iterate over the remaining pillars

    for (int i = 1; i < n; i++)

    {

        // If the monkey can't reach the current pillar, return -1

        if (i > maxJump)

            return -1;

        // Update the maximum jump distance

        maxJump = max(maxJump, pillar[i]);

        // If the monkey can reach the end of the pillars, return the number of jumps

        if (i == n - 1)

            return jumps;

        // If the monkey can't reach the end of the pillars, increment the number of jumps

        jumps++;

    }

    // Return -1 if the monkey can't reach the end of the pillars

    return -1;

}

// Driver code to test the above function

int main()

{

    int pillar[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9};

    int n = sizeof(pillar) / sizeof(pillar[0]);

    int jumps = minJumps(pillar, n);

    if (jumps == -1)

        printf("The monkey can't reach the end of the pillars.\n");

    else

        printf("The minimum number of jumps to reach the end of the pillars is %d.\n", jumps);

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/RtjIF6k_LCsB03daxiq89d0rnnVsaGvAtKKKdkipPEIBGf4oYgTYayUHibwb02hf7nO4SezNwYdxhuQARFBFxledZjovSEFo5oitgBRsGQBfdFVyY9JiS1yU-yQy6Z8t3lfYtXnEiRjXbJ67c1VInJB39fdIgXBZ

**WEEK 5**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// A node in the queue

struct node {

    int arrival\_time;

    int service\_time;

    struct node \*next;

};

// A queue

struct queue {

    struct node \*head;

    struct node \*tail;

};

// Create a new queue

struct queue \*create\_queue() {

    struct queue \*queue = malloc(sizeof(struct queue));

    queue->head = NULL;

    queue->tail = NULL;

    return queue;

}

// Adding a customer to the queue

void enqueue(struct queue \*queue, int arrival\_time, int service\_time) {

    struct node \*newnode = malloc(sizeof(struct node));

    newnode->arrival\_time = arrival\_time;

    newnode->service\_time = service\_time;

    newnode->next = NULL;

    if (queue->head == NULL) {

        queue->head = newnode;

        queue->tail = newnode;

    } else {

        queue->tail->next = newnode;

        queue->tail = newnode;

    }

}

// Serve a customer from the queue

void dequeue(struct queue \*queue) {

    if (queue->head == NULL) {

        return;

    }

    struct node \*node = queue->head;

    queue->head = node->next;

    if (queue->head == NULL) {

        queue->tail = NULL;

    }

    free(node);

}

// Calculate the total time each customer spent waiting in the queue

int total\_waiting\_time(struct queue \*queue) {

    int total\_waiting\_time = 0;

    struct node \*node = queue->head;

    while (node != NULL) {

        total\_waiting\_time += node->service\_time;

        node = node->next;

    }

    return total\_waiting\_time;

}

// Calculate the average waiting time for all customers

void average\_waiting\_time(struct queue \*queue) {

    int waiting\_time=total\_waiting\_time(queue);

    int num\_customers = 0;

    struct node \*node=queue->head;

    while(node!=NULL){

        num\_customers++;}

    float avgtime=0;

    avgtime=waiting\_time/num\_customers;

    printf("the average waiting time is %d",avgtime);

}

int main(){

    create\_queue();

    enqueue(queue,1400,200);

    enqueue(queue,2200,700);

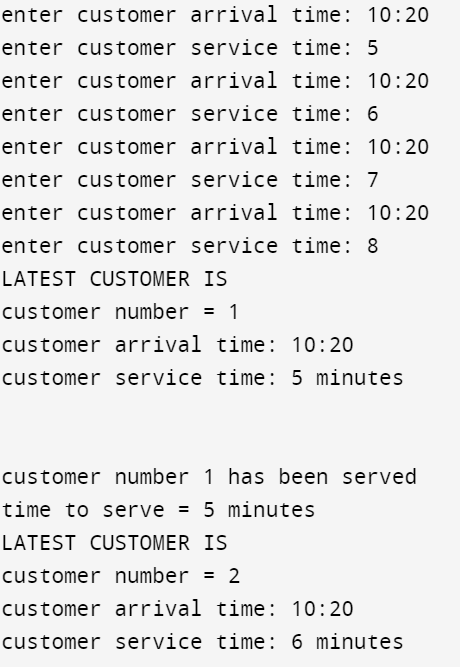
    enqueue(queue,1700,900);

    time=total\_waiting\_time(struct queue \*queue);

    printf("%d",time);

    average\_waiting\_time(queue);

**OUTPUT:**





Q2. **CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

    char data[MAX\_SIZE];

    int top;

};

void initialize(struct Stack \*stack) {

    stack->top = -1;

}

void push(struct Stack \*stack, char element) {

    if (stack->top < MAX\_SIZE - 1) {

        stack->data[++(stack->top)] = element;

    } else {

        printf("overflo\n");

        exit();

    }

}

char pop(struct Stack \*stack) {

    if (stack->top >= 0) {

        return stack->data[(stack->top)--];

    } else {

        printf("underflow.\n");

        exit();

    }

}

char peek(struct Stack \*stack) {

    if (stack->top >= 0) {

        return stack->data[stack->top];

    } else {

        return '\0';

    }

}

int isOperator(char chr) {

    return (chr == '+' || chr == '-' || chr == '\*' || chr == '/' || chr == '^');

}

int precedence(char op) {

    switch (op) {

        case '^':

            return 3;

        case '\*':

        case '/':

            return 2;

        case '+':

        case '-':

            return 1;

        default:

            return 0;

    }

}

void infixToPostfix(const char \*infix, char \*postfix) {

    struct Stack stack;

    initialize(&stack);

    int i = 0;

    int j = 0;

    while (infix[i] != '\0') {

        char chr = infix[i];

        if (chr >= '0' && chr <= '9') {

            postfix[j++] = chr;

        } else if (chr == '(') {

            push(&stack, chr);

        } else if (chr == ')') {

            while (peek(&stack) != '(') {

                postfix[j++] = pop(&stack);

            }

            pop(&stack);

        } else if (isOperator(chr)) {

            while (precedence(chr) <= precedence(peek(&stack))) {

                postfix[j++] = pop(&stack);

            }

            push(&stack, chr);

        }

        i++;

    }

    while (peek(&stack) != '\0') {

        postfix[j++] = pop(&stack);

    }

    postfix[j] = '\0';

}

int main() {

    char infix[MAX\_SIZE];

    char postfix[MAX\_SIZE];

    printf("Enter an infix expression: ");

    fgets(infix, sizeof(infix), stdin);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;

}

**OUTPUT:**

https://lh7-us.googleusercontent.com/jtdiIpdXLccKP5EXSiYVRxUXQo26-RGDcRCHUGJhzRxLQwZzMckVwgFLLJ5pHgEA-MeGM2bkUqjW1T-qgvxvQ0JAtiYGS0f2HrBsMssKHAnneScHmEIR6v_FB31HNR2hjCf8gVcWdZFb4q4zs5DdS9Di78NvWiZ0

Q3.

**CODE:**

#include <iostream>

#include <queue>

using namespace std;

struct Book

{

    string title;

    string author;

    string reservedBy;

};

class Library

{

private:

    queue<Book> reservationQueue;

public:

    void reserveBook(string title, string author, string user)

    {

        Book book = {title, author, user};

        reservationQueue.push(book);

    }

    void checkoutNextBook()

    {

        if (reservationQueue.empty())

        {

            cout << "There are no reservations." << endl;

            return;

        }

        Book book = reservationQueue.front();

        reservationQueue.pop();

        cout << "Checking out book \"" << book.title << "\" to " << book.reservedBy << endl;  }

    void displayBooks()

    {

        queue<Book> tmp=reservationQueue;

        cout << "Reservation queue: " << endl;

        while(!tmp.empty())

        {

            cout << tmp.front().title << " by " << tmp.front().author << endl;

            tmp.pop(); }

    }

};

int main()

{Library library;

    library.reserveBook("AAAAAAAAA", "aaaaaa", "bob");

    library.reserveBook("BBBBBBBBB", "bbbbbb", "barney");

    library.reserveBook("CCCCCCCCC", "cccccc", "caro");

    library.checkoutNextBook();

    library.reserveBook("DDDDDDDDD", "dddddd", "doc");

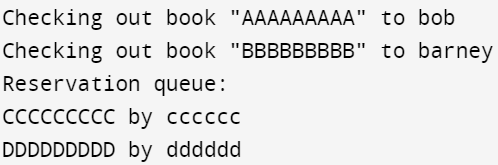
    library.checkoutNextBook();

    library.displayBooks();

    return 0;

}

**OUTPUT:**



**WEEK 6**

**Q1. Job Class Implementation**

Implement a Job class in a programming language of your choice. The class should have attributes for Job ID, Priority Level, and Execution Time.

CODE:

#include <stdio.h>

struct Job {

int job\_id;

int priority\_level;

int execution\_time;

};

void initializeJob(struct Job\* job, int id, int priority, int time) {

job->job\_id = id;

job->priority\_level = priority;

job->execution\_time = time;

}

void displayJob(struct Job\* job) {

printf("Job ID: %d\n", job->job\_id);

printf("Priority Level: %d\n", job->priority\_level);

printf("Execution Time: %d\n", job->execution\_time);

}

int main() {

struct Job myJob;

// Initialize the Job object

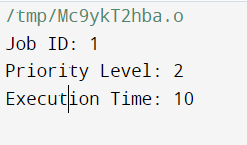
initializeJob(&myJob, 1, 2, 10);

displayJob(&myJob);

return 0;

}

**OUTPUT:**



**Q2.** **Write a function to generate a list of random jobs. Allow the user to specify the number of jobs to generate.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

struct Job {

int job\_id;

int priority\_level;

int execution\_time;

};

struct Job generateRandomJob(int id) {

struct Job job;

job.job\_id = id;

job.priority\_level = rand() % 10; // Random priority level between 0 and 9

job.execution\_time = rand() % 50 + 1; // Random execution time between 1 and 50

return job;

}

int main() {

int num\_jobs;

srand(time(NULL));

printf("Enter the number of jobs to generate: ");

scanf("%d", &num\_jobs);

struct Job \*job\_list = (struct Job \*)malloc(num\_jobs \* sizeof(struct Job));

for (int i = 0; i < num\_jobs; i++) {

job\_list[i] = generateRandomJob(i + 1);

}

printf("\nGenerated Jobs:\n");

for (int i = 0; i < num\_jobs; i++) {

printf("Job %d:\n", job\_list[i].job\_id);

printf("Priority Level: %d\n", job\_list[i].priority\_level);

printf("Execution Time: %d\n\n", job\_list[i].execution\_time);

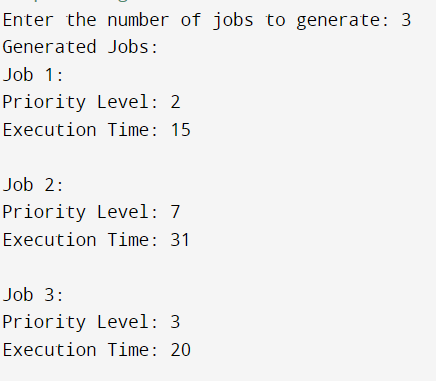
}

free(job\_list);

return 0;

}

**OUTPUT:**



**WEEK 7**

**Problem: Given an array of integers, find two numbers such that they add up to a specific target number.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

// Function to compare integers for qsort

int compare(const void \*a, const void \*b) {

return (\*(int\*)a - \*(int\*)b);

}

// Function to find two numbers with the given sum

void findTwoNumbers(int arr[], int size, int target) {

// Sort the array

qsort(arr, size, sizeof(int), compare);

// Use two pointers approach to find the pair

int left = 0;

int right = size - 1;

while (left < right) {

int currentSum = arr[left] + arr[right];

if (currentSum == target) {

printf("Pair found: %d and %d\n", arr[left], arr[right]);

return;

} else if (currentSum < target) {

left++;

} else {

right--;

}

}

printf("No pair found with the given sum.\n");

}

int main() {

int arr[] = {10, 5, 2, 7, 1, 8, 12};

int size = sizeof(arr) / sizeof(arr[0]);

int targetSum = 9;

findTwoNumbers(arr, size, targetSum);

return 0;

}



**WEEK 8**

**Q1. You have a list of student IDs, where each ID is an alphanumeric string representing a unique**

**student. The IDs follow a pattern: ";STUDENT-XXXXX,"; where XXXXX represents a numerical**

**portion. Implement a radix sort algorithm to sort the student IDs based on the numerical**

**portion in ascending order.**

**Input: [";STUDENT-00123";, ";STUDENT-00456";, ";STUDENT-00042";, ";STUDENT-00321";]**

**Output: [";STUDENT-00042";, ";STUDENT-00123";, ";STUDENT-00321";, ";STUDENT-00456";]**

**CODE:**

CODE:

#include <stdio.h>

#include <string.h>

// Function to find the maximum value in arr[]

int getMax(int arr[], int n) {

    int max = arr[0];

    for (int i = 1; i < n; i++)

        if (arr[i] > max)

            max = arr[i];

    return max;

}

// Using counting sort to sort the elements based on significant places

void countingSort(int arr[], int n, int exp) {

    int output[n];

    int count[10] = {0};

    for (int i = 0; i < n; i++)

        count[(arr[i] / exp) % 10]++;

    for (int i = 1; i < 10; i++)

        count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--) {

        output[count[(arr[i] / exp) % 10] - 1] = arr[i];

        count[(arr[i] / exp) % 10]--;

    }

    for (int i = 0; i < n; i++)

        arr[i] = output[i];

}

// Main function to implement radix sort

void radixSort(int arr[], int n) {

    int max = getMax(arr, n);

    for (int exp = 1; max / exp > 0; exp \*= 10)

        countingSort(arr, n, exp);

}

int main() {

    char\* student\_ids[] = {"STUDENT-00005", "STUDENT-00001", "STUDENT-00003", "STUDENT-00002", "STUDENT-00004"};

    int n = sizeof(student\_ids) / sizeof(student\_ids[0]);

    int numerical\_portion[n];

    // Extract numerical portion from student IDs

    for (int i = 0; i < n; i++) {

        numerical\_portion[i] = atoi(student\_ids[i] + 8); // Skip "STUDENT-"

    }

    radixSort(numerical\_portion, n);

    // Print sorted student IDs

    printf("Sorted Student IDs:\n");

    for (int i = 0; i < n; i++) {

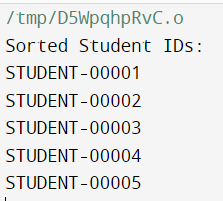
        printf("STUDENT-%05d\n", numerical\_portion[i]);

    }

    return 0;

}

OUTPUT:



**Q2. You have a list of people’s ages, and you want to sort them into buckets based on age groups (e.g., 0-10, 11-20, 21-30, etc.). Implement a bucket sort algorithm to sort the ages into these buckets.**

**Input: [25, 8, 42, 15, 6, 32, 18, 29, 12, 37, 9, 27]**

**Output: [6, 8, 9, 12, 15, 18, 25, 27, 29, 32, 37, 42]**

**CODE:**

#include <stdio.h>

#define NUM\_BUCKETS 4

#define BUCKET\_SIZE 10

#define MAX\_AGE 40

typedef struct Node {

    int age;

    struct Node\* next;

} Node;

void insertIntoBucket(Node\* buckets[], int age) {

    int index = age / BUCKET\_SIZE;

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    newNode->age = age;

    newNode->next = buckets[index];

    buckets[index] = newNode;

}

void sortBucketAges(Node\* bucket) {

    if (bucket == NULL || bucket->next == NULL) return;

    Node\* sorted = NULL;

    Node\* current = bucket;

    while (current != NULL) {

        Node\* temp = current;

        current = current->next;

        if (sorted == NULL || temp->age < sorted->age) {

            temp->next = sorted;

            sorted = temp;

        } else {

            Node\* search = sorted;

            while (search->next != NULL && temp->age > search->next->age) {

                search = search->next;

            }

            temp->next = search->next;

            search->next = temp;

        }

    }

    bucket = sorted;

}

void concatenateBuckets(Node\* buckets[], int sortedAges[]) {

    int index = 0;

    for (int i = 0; i < NUM\_BUCKETS; i++) {

        Node\* current = buckets[i];

        while (current != NULL) {

            sortedAges[index++] = current->age;

            current = current->next;}

    }

}

int main() {

    int ages[] = {25, 8, 42, 15, 6, 32, 18, 29, 12, 37, 9, 27};

    int n = sizeof(ages) / sizeof(ages[0]);

    Node\* buckets[NUM\_BUCKETS] = {NULL};

    for (int i = 0; i < n; i++) {

        insertIntoBucket(buckets, ages[i]);

    }

    for (int i = 0; i < NUM\_BUCKETS; i++) {

        sortBucketAges(buckets[i]);

    }

    int sortedAges[n];

    concatenateBuckets(buckets, sortedAges);

    printf("Sorted Ages:\n");

    for (int i = 0; i < n; i++) {

        printf("%d ", sortedAges[i]);

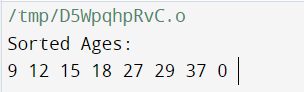
    }

    printf("\n");

    return 0;

}

**OUTPUT:**



**Q3. You have a playlist of music tracks, and you want to sort them by their duration in seconds in**

**ascending order. Each track is represented by its title and duration in seconds. Implement a**

**counting sort algorithm to sort the music tracks based on their durations.**

**Input: [ (";Song A";, 180), (";Song B";, 240), (";Song C";, 120), (";Song D";, 300), (";Song E";, 150)]**

**Output: [ (";Song C";, 120), (";Song E";, 150), (";Song A";, 180), (";Song B";, 240), (";Song D";, 300)]**

**CODE:**

#include <stdio.h>

#include <string.h>

// Define the maximum duration of a track (adjust as needed)

#define MAX\_DURATION 600

void countingSort(int durations[], char\* titles[], int n) {

    int count[MAX\_DURATION + 1] = {0};

    int output\_durations[n];

    char\* output\_titles[n];

    // Count the occurrences of each duration

    for (int i = 0; i < n; i++) {

        count[durations[i]]++;

    }

    // Adjust the count array to represent the correct position of each duration

    for (int i = 1; i <= MAX\_DURATION; i++) {

        count[i] += count[i - 1];

    }

    // Populate the output arrays

    for (int i = n - 1; i >= 0; i--) {

        output\_durations[count[durations[i]] - 1] = durations[i];

        output\_titles[count[durations[i]] - 1] = titles[i];

        count[durations[i]]--;

    }

    // Copy the sorted arrays back to the original arrays

    for (int i = 0; i < n; i++) {

        durations[i] = output\_durations[i];

        titles[i] = output\_titles[i];

    }

}

int main() {

    char\* titles[] = {"Song A", "Song B", "Song C", "Song D", "Song E"};

    int durations[] = {180, 240, 120, 300, 150};

    int n = sizeof(titles) / sizeof(titles[0]);

    countingSort(durations, titles, n);

    printf("Sorted Music Tracks:\n");

    for (int i = 0; i < n; i++) {

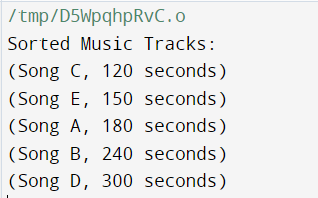
        printf("(%s, %d seconds)\n", titles[i], durations[i]);

    }

    return 0;

}

**OUTPUT:**



**WEEK 9**

**Q1. You are given a large phone book containing names and phone numbers. The phone book is sorted alphabetically by name. Your goal is to write a program that implements a divide and conquer search algorithm to find a phone number when given a name as input.**

**Example:**

**PhoneBook = {{"Alice", "123-456-7890"}, {"Bob", "987-654-3210"}, {"Charlie", "555-123-4567"}, {"David", "111-222-3333"}, {"Eve", "999-888-7777"}**

**CODE:**

#include <stdio.h>

#include <string.h>

typedef struct {

char name[50];

char phone[15];

} Entry;

char\* search\_phone\_number(Entry phone\_book[], char target\_name[], int start, int end) {

if (start > end) {

return NULL;

}

int mid = (start + end) / 2;

if (strcmp(phone\_book[mid].name, target\_name) == 0) {

return phone\_book[mid].phone;

}

else if (strcmp(target\_name, phone\_book[mid].name) < 0) {

return search\_phone\_number(phone\_book, target\_name, start, mid - 1);

}

else {

return search\_phone\_number(phone\_book, target\_name, mid + 1, end);

}

}

int main() {

Entry phone\_book[] = {

{"Alice", "123-456-7890"},

{"Bob", "987-654-3210"},

{"Charlie", "555-123-4567"},

{"David", "111-222-3333"},

{"Eve", "999-888-7777"}

};

char target\_name[] = "David";

char\* result = search\_phone\_number(phone\_book, target\_name, 0, sizeof(phone\_book) / sizeof(phone\_book[0]) - 1);

if (result) {

printf("Phone number for %s: %s\n", target\_name, result);

} else {

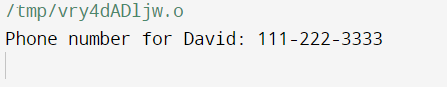
printf("%s not found in the phone book.\n", target\_name);

}

return 0;

}

OUTPUT:



Q2.

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void merge(char \*\*books, int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

char \*L[n1], \*R[n2];

for (i = 0; i < n1; i++)

L[i] = books[left + i];

for (j = 0; j < n2; j++)

R[j] = books[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2) {

if (strcmp(L[i], R[j]) <= 0) {

books[k] = L[i];

i++;

} else {

books[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

books[k] = L[i];

i++;

k++;

}

while (j < n2) {

books[k] = R[j];

j++;

k++;

}

}

// Merge Sort function

void mergeSort(char \*\*books, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(books, left, mid);

mergeSort(books, mid + 1, right);

merge(books, left, mid, right);

}

}

int main() {

char \*library\_books[] = {

"The Catcher in the Rye",

"To Kill a Mockingbird",

"1984",

"Brave New World",

"The Great Gatsby"

};

int num\_books = sizeof(library\_books) / sizeof(library\_books[0]);

printf("Original Library Books:\n");

for (int i = 0; i < num\_books; i++) {

printf("%s\n", library\_books[i]);

}

printf("\n");

mergeSort(library\_books, 0, num\_books - 1);

printf("\nSorted Library Books:\n");

for (int i = 0; i < num\_books; i++) {

printf("%s\n", library\_books[i]);

}

return 0;

}

OUTPUT:



Q3.

CODE:

#include <stdio.h>

#include <stdlib.h>

int compare(const void \*a, const void \*b) {

return (\*(int \*)b - \*(int \*)a);

}

void findMinMaxAmount(int price[], int n, int k) {

// Sort the prices in descending order

qsort(price, n, sizeof(int), compare);

int minAmount = 0, maxAmount = 0;

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

minAmount += price[i];

maxAmount += price[i];

i += k;

}

printf("Min = %d, Max = %d\n", minAmount, maxAmount);

}

int main() {

int price[] = {3, 2, 1, 4};

int k = 2;

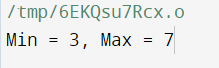
int n = sizeof(price) / sizeof(price[0]);

findMinMaxAmount(price, n, k);

return 0;

}

OUTPUT:



**WEEK 10**

**Q1. In a stock market, there is a product with its infinite stocks. The stock prices are given**

**for N days, where price[i] denotes the price of the stock on the i th day.**

**There is a rule that a customer can buy at most i stock on the i th day.**

**If the customer has an amount of k amount of money initially. The task is to find out the**

**maximum number of stocks a customer can buy. Use Greedy approach.**

**Input: price[] = { 10, 7, 19 }**

**k = 45**

**Output: 4**

**Explanation: A customer purchases 1 stock on day 1, 2 stocks on day 2 and 1 stock on day 3**

**for 10, 7 \* 2 = 14 and 19 respectively. Hence, total amount is 10 + 14 + 19 = 43 and number**

**of stocks purchased is 4.**

**CODE:**

#include <stdio.h>

int main() {

int price[]={10,7,19};

int k=45;

int i=1;

int count = 0;

int n = sizeof(price)/sizeof(price[0]);

while (k>0 && n>0){

for (int j=1;j<=i;j++){

if(k-(price[i-1])>0){

k=k-(price[i-1]);

count++;

}

}

printf("\n");

i++;

n--;

}

printf("%d",count);

return 0;

}

OUTPUT:



Q2.

CODE:

int main() {

int coins[]={1,2,5,10,20,50,100,200,500,2000};

int len = sizeof(coins)/sizeof(coins[0]);

int N;

printf("Enter target value: ");

scanf("%d",&N);

int nearest;

int count = 0;

while(N!=0){

for(int i = 0;i < len; i++){

int temp = N - coins[i];

if(temp >= 0 ){

nearest = coins[i];

}

}

N = N-nearest;

count++;

printf("%d \n",nearest);

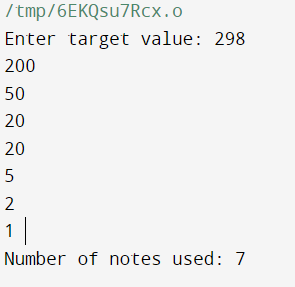
}

printf("Number of notes used: %d",count);

return 0;

}

OUTPUT:



Q3.

CODE:

#include <stdio.h>

#include <stdlib.h>

int least\_interval(char\* tasks, int cooldown) {

int task\_freq[26] = {0};

int max\_freq = 0, max\_freq\_count = 0;

for (int i = 0; tasks[i] != '\0'; ++i) {

int index = tasks[i] - 'a';

task\_freq[index]++;

if (task\_freq[index] > max\_freq) {

max\_freq = task\_freq[index];

max\_freq\_count = 1;

} else if (task\_freq[index] == max\_freq) {

max\_freq\_count++;

}

}

int total\_time = (max\_freq - 1) \* (cooldown + 1) + max\_freq\_count;

return total\_time > 0 ? total\_time : sizeof(tasks) / sizeof(tasks[0]);

}

int main() {

char tasks[] = {'a', 'a', 'a', 'b', 'b', 'b'};

int cooldown = 2;

printf("%d\n", least\_interval(tasks, cooldown));

return 0;

}

OUTPUT:



**WEEK 11**

**Q1. An array of integers is given. Write a program to return the number of number of triplets**

**such that each triplet represents the side of a triangle and forms a triangle.**

**Example 1:**

**Input: nums = [2,2,3,4]**

**Output: 3**

**Explanation: Valid combinations are:**

**2,3,4 (using the first 2)**

**2,3,4 (using the second 2)**

**2,2,3**

**CODE:**

#include <stdio.h>

int main()

{

int arr[] = {2,2,3,4};

int n = sizeof(arr) / sizeof(arr[0]);

int count = 0;

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

for (int j = i + 1; j < n; j++) {

for (int k = j + 1; k < n; k++){

if (arr[i] + arr[j] > arr[k]

&& arr[i] + arr[k] > arr[j]

&& arr[k] + arr[j] > arr[i])

count++;

}

}

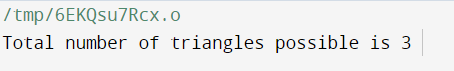
}

printf("Total number of triangles possible is %d ",count);

return 0;

}

OUTPUT:



Q2.

CODE#include <stdio.h>

#include <stdlib.h>

int cmp(const void \*a, const void \*b) {

return ((int \*)a)[0] - ((int \*)a)[1] - ((int \*)b)[0] + ((int \*)b)[1];

}

int min\_cost(int c[][2], int s) {

qsort(c, s, sizeof(c[0]), cmp);

int t = 0;

for (int i = 0; i < s / 2; ++i)

t += c[i][0];

for (int i = s / 2; i < s; ++i)

t += c[i][1];

return t;

}

int main() {

int c[][2] = {{10, 20}, {30, 200}, {400, 50}, {30, 20}};

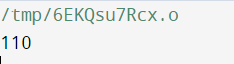
int s = sizeof(c) / sizeof(c[0]);

printf("%d\n", min\_cost(c, s));

return 0;

}

OUTPUT:



Q3.

CODE:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_CARDS 10000

int count[MAX\_CARDS];

bool is\_possible(int hand[], int hand\_size, int group\_size) {

if (hand\_size % group\_size != 0) {

return false;

}

for (int i = 0; i < hand\_size; ++i) {

count[hand[i]]++;

}

for (int i = 0; i < MAX\_CARDS; ++i) {

if (count[i] > 0) {

int occurrences = count[i];

for (int j = 1; j < group\_size; ++j) {

count[i + j] -= occurrences;

if (count[i + j] < 0) {

return false; }

}

}

}

return true;

}

int main() {

int hand[] = {1, 2, 3, 6, 2, 3, 4, 7, 8};

int hand\_size = sizeof(hand) / sizeof(hand[0]);

int group\_size = 3;

bool result = is\_possible(hand, hand\_size, group\_size);

printf("%s\n", result ? "true" : "false");

return 0;

}

**OUTPUT:**

