

Instructions: Please write each program from scratch on your own. No copying and pasting from slides or use of tools like ChatGPT should be deployed. You must carefully look into the class materials before attempting and ask for help from your designated TA if you fail after a sincere effort.

You are discouraged from talking loudly in the lab, as others may feel disturbed. You may discuss with your neighboring batchmate, but in a soft tone. You are also discouraged from using the lab computers for other activities except solving the assignment during the lab session.

Set 1

1. Write a program in C programming language to print your name on the terminal.
2. Write a program in C programming language to take a number as input from the user and print its square on the terminal.
3. Write a program in C programming language to take as input two numbers and swap them using a third variable.
4. Write a program in C programming language to take as input two numbers and swap them without using a third variable.

```
#include<stdio.h>
// Program in C programming language to take
//as input two numbers and swap them without using a third variable

void main()
{
    int a, b;
    printf("Give two numbers as input:\n");
    scanf("%d %d", &a, &b); //a =5, b = 3
    printf("The numbers before swap ==> a = %d, b = %d\n\n", a, b);
    a = a + b; //a = 8, b = 3
    b = a - b; //a = 8, b = 5
    a = a - b; //a = 3, b = 5

    printf("The numbers after swap ==> a = %d, b = %d\n\n", a, b);
}
```

5. Write a program in C programming language to take as input the base (b) and height (h) of a triangle and print the area on the terminal (formula: $\text{area} = (\frac{1}{2}) * \text{base} * \text{height}$).
6. Write a program in C programming language to take the principal amount (P), time in years (T), and rate of interest (R) as input and print the Simple Interest (SI) and the Total Amount (A) on the terminal (formula: $\text{SI} = (\text{P} * \text{T} * \text{R}) / 100$, $\text{A} = \text{P} + \text{SI}$).

```
//C program to take the principal amount (P), time in years (T),  
//and rate of interest ( R ) as input  
//and print the Simple Interest (SI) and the Total Amount (A) on the terminal
```

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

```
void main()  
{  
    float P, T, R, SI, A;  
    printf("Give principal amount, time in years and rate of interest below:\n");  
    scanf("%f %f %f", &P, &T, &R);  
    SI = (P*T*R)/100;  
    A = P + SI;  
    printf("The Simple Interest is %f, the total amount is %f", SI, A);  
}
```

7. Write a program in C programming language to take as input the radius (R) and height (H) of a cone and print the volume on the terminal (formula: $\text{volume} = \text{PI} * \text{r} * \text{r} * (\text{h} / 3)$).
8. Write a C program to take the name (e.g., Kripa) and age (41) (supposing your name is "Sachin" and your age is 50) as input from the user and print a message on the terminal: "Hello Kripa, you are 41 years old. I am Sachin, I am 50 years old".

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

```
void main()  
{  
    char name[25];  
    int age;  
    printf("Give your name: ");  
    scanf("%s", name);  
    printf("\nGive your age: ");  
    scanf("%d", &age);
```

```
printf("Hello %s, you are %d years old. I am Sachin, I am 50 years old", name,
age);
}
```

9. Correct the following C programs and run the corrected versions:

i)

//Desired output: Hello world!\n

```
int main()
{
    print("Hello world!\n")
}
```

#include<stdio.h>

```
int main()
{
    printf("Hello world!\n");
}
```

ii)

//Make sure the program runs. The value of k is not important.

#include<stdio.h>

```
int main()
{
    int i = 5, j = 6, k;

    k = (i + j)/(j - 6);

    printf("k = %d\n", k);
}
```

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

```
int main()
```

```
{
```

```
    int i = 5, j = 6, k;
```

```
    k = (i + j)/(j - 5);
```

```
    printf("k = %d\n", k);
```

```
}
```

iii)

//Desired output: Miss mona please transfer Rs. 25000 to my account.

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

```
int main()
```

```
{
```

```
    int amount = 25,000;
```

```
    char name[] = 'mona';
```

```
    printf("Miss %s please transfer Rs. %d to my account.\n", name, amount);
```

```
}
```

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

```
int main()
```

```
{
```

```
    int amount = 25000;
```

```
    char name[] = "mona";
```

```
    printf("Miss %s please transfer Rs. %d to my account.\n", name, amount);
```

```
}
```

Set 2

1. Write a program in C to calculate the Gravitation Force (F) between two objects of masses M1 and M2 (Formula: $F = (G \cdot M1 \cdot M2) / (R \cdot R)$, $G = 6.673 \times 10^{-11}$, R is the distance between the two centres of masses of the two objects) such that G is i) a macro (#define), ii) local variable, iii) global variable.

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

//#define G 6.673*10E-11
float G = 6.673*10E-11;

void main()
{
    float F, M1 = 1000, M2 = 2000, R = 50;
    float G = 6.673*10E-11;

    F = (G*M1*M2)/(R*R);
    printf("F = %.20f\n", F);
}
```

2. Name functions as RollaCosta, Dominos, and Bawarchi – three restaurants in Kalyani. Every time you call a function, a message will be printed indicating the visit number of that function (restaurant). E.g., if you call RollaCosta, it will print “Welcome to Rolla Costa! This is your visit number 2” (assuming this is your 2nd call of the function). Implement this scheme using C programming language by using i) global variables and ii) static variables.

i) Global variables

```
#include<stdio.h>

int visit_rollacosta = 0;
int visit_dominos = 0;
int visit_bawarchi = 0;

void RollaCosta()
{
```

```

        visit_rollacosta++;
        printf("Welcome to Rolla Costa! This is your visit number %d\n\n",
visit_rollacosta);
    }

void Dominos()
{

        visit_dominos++;
        printf("Welcome to Dominos! This is your visit number %d\n\n",
visit_dominos);
    }

void Bawarchi()
{

        visit_bawarchi++;
        printf("Welcome to Bawarchi! This is your visit number %d\n\n",
visit_bawarchi);
    }

void main()
{
        RollaCosta();
        RollaCosta();
        RollaCosta();

        Dominos();
        Dominos();
        Dominos();

        Bawarchi();
        Bawarchi();
        Bawarchi();

}

```

ii) Static variables

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

```

void RollaCosta()
{
    int static visit = 0;
    visit++;
    printf("Welcome to Rolla Costa! This is your visit number %d\n\n", visit);
}

void Dominos()
{
    int static visit = 0;
    visit++;
    printf("Welcome to Dominos! This is your visit number %d\n\n", visit);
}

void Bawarchi()
{
    int static visit = 0;
    visit++;
    printf("Welcome to Bawarchi! This is your visit number %d\n\n", visit);
}

void main()
{
    RollaCosta();
    RollaCosta();
    RollaCosta();

    Dominos();
    Dominos();
    Dominos();

    Bawarchi();
    Bawarchi();
    Bawarchi();

}

```

3. Consider a secret agent service with four secret agents: 002, 003, 005, and 007. The agents do not know each other, and they interact directly with the head of the organization (main()). The head interacts (function calls) with each agent separately, and in response, the agent prints his/her passcode (on the console).

The passcode of agent 00i ($i = 2, 3, 5, 7$) is an integer divisible by i . Right after the response, each agent updates the passcode such that it remains divisible by i (the updation scheme is known only to the agent and the head). There can be multiple such interactions. Note that the passcode of one agent is not visible to any other agent. The head can only view it upon the interaction (after which it is updated by the agent), but can not access it directly (the head knows that it is divisible by i for a sanity check, but does not know it exactly before the interaction). Using C and an appropriate storage class (variable type), implement this scheme. You may choose your own initial passcode for each agent i such that it is divisible by i .

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

```
void agent2()
```

```
{  
    static int passcode = 1234;  
  
    printf("Agent 002 reporting! Passcode = %d\n", passcode);  
    passcode += 2;  
}
```

```
void agent3()
```

```
{  
    static int passcode = 1233;  
  
    printf("Agent 003 reporting! Passcode = %d\n", passcode);  
    passcode += 3;  
}
```

```
void agent5()
```

```
{  
    static int passcode = 1235;  
  
    printf("Agent 005 reporting! Passcode = %d\n", passcode);  
    passcode += 5;  
}
```

```
void agent7()
```

```
{  
    static int passcode = 1239;  
  
    printf("Agent 007 reporting! Passcode = %d\n", passcode);  
    passcode += 7;  
}
```



```

void main()
{
    agent2();
    agent3();
    agent5();
    agent7();

    agent2();
    agent3();
    agent5();
    agent7();

    agent2();
    agent3();
    agent5();
    agent7();

    agent2();
    agent3();
    agent5();
    agent7();
}

```

4. Write a program to simulate the updation process of the current stock for three books 1, 2, and 3 handled by a librarian through three functions updateStockbook_i (i = 1, 2, 3) using a) global variables, b) static variables. The update is an increment of 1 in the stock of that book at a time. Print the current stock within the corresponding function after the update.

```

#include<stdio.h>

void updateStockbook_1()
{
    static int book1 = 0;

    book1++;

    printf("The current stock of book 1 is %d.\n", book1);
}

```

```

void updateStockbook_2()
{
    static int book2 = 0;

    book2++;

    printf("The current stock of book 2 is %d.\n", book2);

}

void updateStockbook_3()
{
    static int book3 = 0;

    book3++;

    printf("The current stock of book 3 is %d.\n", book3);

}

int main()
{
    updateStockbook_1();
    updateStockbook_2();
    updateStockbook_3();

    updateStockbook_1();
    updateStockbook_2();
    updateStockbook_3();

    updateStockbook_1();
    updateStockbook_2();
    updateStockbook_3();

}

```

5. Identify the issue in the following code and resolve it, ensuring none of the existing variables are removed and no line is commented out.

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

```
int main() {  
  
    int x = 10;  
  
    { //Block 1  
        int y = 20;  
        { //Block2  
            int x = 99;  
  
            printf("Sum1: %d\n", x + y);  
        }  
  
        printf("Sum2: %d\n", x + y);  
    }  
  
    printf("Sum3: %d\n", x + y);  
  
    return 0;  
}
```

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

```
int main() {  
  
    int x = 10;  
  
    { //Block 1  
        int y = 20;  
        { //Block2  
            int x = 99;  
  
            printf("Sum1: %d\n", x + y);  
        }  
  
        printf("Sum2: %d\n", x + y);  
    }  
  
    int y = 5;
```

```

printf("Sum3: %d\n", x + y);

return 0;
}

```

Set 3

1. Write a program to print the absolute value of an integer (I), which is the negation of the integer I if I is negative, else it is the same as I. Apply both if-else and the conditional operator (? :). There is no need to take user input.

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

```

int main() {
    int num = -5;
    int abs_value = num;;

    if(num < 0)
    {
        abs_value = -num;
    }

    printf("Absolute value of %d is %d\n", num, abs_value);
    return 0;
}

```

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

```

int main() {
    int num = -5;
    int abs_value = (num < 0) ? -num : num;
    printf("Absolute value of %d is %d\n", num, abs_value);
    return 0;
}

```

2. Write a program to print the floor and ceil of a real number **without** using ceil() and floor() functions of math.h (or any other in-built functions). E.g., for a number n = 55.5, floor(n) = 55.0 (i.e., the immediately **previous** integer) and ceil(n) = 56.0 i.e., the immediately **next** integer). There is no need to take user input.

```

#include<stdio.h>

void main()
{
    int c, f;
    float n = 55.00001, fract_part;

    f = (int)n;//floor
    c = n;//ceil

    fract_part = n - f;

    if(fract_part > 0)
    {
        c = n + 1;
    }

    printf("ceil(%f) = %d floor(%f) = %d\n", n, c, n, f);
}

```

3. Write a program to find the maximum and the second maximum of three integers (taken as input from the terminal) using if-else.

```

#include<stdio.h>

void main()
{
    int a = 30, b = 40, c = 5, max, max2;

    //max = a;

    if(a > b)//a > b
    {

        if(a > c)
        {
            max = a;
            if(b > c)

```

```

        {
            max2 = b;
        }
        else
        {
            max2 = c;
        }
    }
    else//a <= c; a > b
    {
        max = c;//c > a; a > b ==> c > b
        max2 = a;//a>b
    }
}
else//b > a
{
    if(b > c)
    {
        max = b;
        if(a > c)
        {
            max2 = a;
        }
        else
        {
            max2 = c;
        }
    }
    else//b <= c
    {
        max = c; //b > a; c > b ==> c > a
        max2 = b;
    }
}

printf("Max = %d 2nd max = %d\n", max, max2);
}

```

4. Write a program to calculate the electricity bill based on the units and the rates as given below, with an appropriate decision-making structure (e.g., if-else-if). The units (an

integer) will be taken as input from the user, and the total bill (using “%f”), based on the rate, in Rs. will be printed on the terminal.

Units	Rate (Rs. per unit)
First 25	4.89
Next 35	5.40
Next 40	6.41
Next 50	7.16
Next 50	7.33
Next 100	7.33
Above 300	8.92

- Write a program to assign teams (one for the Females and another for the Males) to students for the annual sports meet based on the criteria below, with an appropriate decision-making structure (e.g., if-else-if). You should ask for the Roll Number (an integer) and Gender (a character – M/F; declared as char and read using “%c”) as input from the terminal and print the Team Name for the student.

Roll number	Gender	Team name
1 - 25	Female	Mary Kom Smashers
26- 50	Female	Smriti Mandana Dashers
> 50	Female	Manu Bhaker Snipers
1 - 20	Male	Virat Kohli Challengers
21 - 40	Male	R Praggnanandhaa Combaters
> 40	Male	Neeraj Chopra Warriors

- Consider a Genie standing in front of the treasure door. The Genie asks you a puzzle, e.g., “The day before two days after the day before tomorrow is Saturday. What day is it today?” with four options like a) Sunday, b) Monday..... The Genie lets you in only if the answer is correct; else imprisons you. Implement this scheme using switch-case, where you print messages based on success, failure, or invalid answer.

Can you also do it using a maximum of three “break”s?

```
#include<stdio.h>

void main()
{
    char option;

    printf("Give the correct option for the puzzle:\n");
    printf("Puzzle: The day before two days after the day before tomorrow is
Saturday. What day is it today?\n\n");
    printf("Options: a) Sunday, b) Monday, c) Friday, d) Thursday.\n\n");
    printf("Please input one of the above four options below:\n");
    scanf("%c", &option);

    switch(option)
    {
        case 'c': printf("Correct! The treasure is yours!\n");
                    break;

        case 'a':

        case 'b':

        case 'd': printf("Wrong! You are imprisoned!\n");
                    break;

        default: printf("Wrong option!!!");
                  break;
    }
}
```

7. An integer n when divided by another integer d (>0) belongs to exactly one of the mod d equivalence classes $(Z_d) \{[0], [1], [2], \dots [d-1]\}$ where $[i]$ (i being an integer) is the set of all integers with remainder i when divided by d such that $i = 0, 1, 2, \dots, d - 1$. That is, if $n = 20$, $d = 5$, n will belong to the class $[0]$; if $n = 21$, $d = 5$, n will belong to $[1]$, and so on. Design a C program to print the mod d equivalent class of n using switch case. Note that it doesn't need to be a generic solution for any d (no need to take 'n' and 'd' as user inputs, and you can have fixed n , d values).


```

#include<stdio.h>

void main()
{
    int n = 21, d = 5;

    switch(n%d)
    {
        case 0:    printf("[0]\n");
                   break;

        case 1:    printf("[1]\n");
                   break;

        case 2:    printf("[2]\n");
                   break;

        case 3:    printf("[3]\n");
                   break;

        case 4:    printf("[4]\n");
                   break;

    }
}

```

8. Write a C program to simulate the reception of an online e-shop. First, greet the customer by printing a welcome message. Then, print the catalog of five items (with a product id like 0, 1,..) with the prices and quantities present for each. Next, request the customer to provide the id of the product s/he want to buy. Using switch-case (that will move to the appropriate 'case' for the product), for each product, print a message asking for the quantity of the product selected (exit with an appropriate message if a wrong option has been provided for the product id). If the quantity is less than or equal to that of the available item, print the message with the final bill and update the stock (if the quantity requested is more than that available, print an appropriate message and exit). Finally, print a nice goodbye message.

```

#include<stdio.h>

void main()
{
    int apple_n = 5, mango_n = 10, banana_n = 50, pineapple_n = 40, papaya_n =
15;
    float apple_p = 10.0, mango_p = 50.0, banana_p = 5.0, pineapple_p = 55.5,
papaya_p = 30.5;
    int code, qty;

    printf("Welcome to Fruitopia!!\n\n");
    printf("Catalogue:\n\n");
    printf("Apple (code 0): quantity = %d price = %f INR per piece\n", apple_n,
apple_p);
    printf("Mango (code 1): quantity = %d price = %f INR per piece\n", mango_n,
mango_p);
    printf("Banana (code 2): quantity = %d price = %f INR per piece\n", banana_n,
banana_p);
    printf("Pineapple (code 3): quantity = %d price = %f INR per piece\n",
pineapple_n, pineapple_p);
    printf("Papaya (code 4): quantity = %d price = %f INR per piece\n", papaya_n,
papaya_p);

    printf("Please give the code of the fruit you want to buy!\n\n");
    scanf("%d", &code);

    switch(code)
    {
        case 0:    //Apple
                    printf("Quantity please:\n");
                    scanf("%d", &qty);
                    if(qty > apple_n)
                    {
                        printf("Sorry! We are out of stock!!\n");
                    }
                    else
                    {
                        printf("Bill = %f INR", qty*apple_p);
                    }
                    apple_n -= qty;
                    break;

        case 1:    //Mango
                    printf("Quantity please:\n");

```

```
scanf("%d", &qty);
if(qty > mango_n)
{
    printf("Sorry! We are out of stock!!\n");
}
else
{
    printf("Bill = %f INR", qty*mango_p);
}
mango_n -= qty;
break;
```

case 2: //Banana

```
printf("Quantity please:\n");
scanf("%d", &qty);
if(qty > banana_n)
{
    printf("Sorry! We are out of stock!!\n");
}
else
{
    printf("Bill = %f INR", qty*banana_p);
}

banana_n -= qty;
break;
```

case 3: //Pineapple

```
printf("Quantity please:\n");
scanf("%d", &qty);
if(qty > pineapple_n)
{
    printf("Sorry! We are out of stock!!\n");
}
else
{
    printf("Bill = %f INR", qty*pineapple_p);
}
pineapple_n -= qty;
break;
```

case 4: //Papaya

```
printf("Quantity please:\n");
scanf("%d", &qty);
```

```

        if(qty > papaya_n)
        {
            printf("Sorry! We are out of stock!!\n");
        }
        else
        {
            printf("Bill = %f INR", qty*papaya_p);
        }
        papaya_n -= qty;
        break;

    default: printf("Invalid choice!");
            break;

}

printf("\nPlease visit again!!\n");

}

```

9. In Computer Science, peer-reviewed international conferences are widely considered to be highly prestigious, and they have a ranking as CORE A* (the highest), A, B, and C (see <https://portal.core.edu.au/conf-ranks/>). As an example, EMNLP is A*, shown below.

ICORE Conference Portal

CORE rankings is now an international collaboration named ICORE

Any rankings questions should be sent to icore.rankings@gmail.com

[ICORE rankings page](#) | [Frequently asked questions](#) | [Description of Conference Ranks](#) | [Fields of Research codes](#)

[CORE Dora discussion document](#) | [Guidelines for reporting rankings](#) | [Usage of CORE rankings](#) | [ICORE addition and upgrade requirements](#) | [Data used in Rankings a](#)

Search by: All

Source: CORE2023

Showing results 1 - 1 of 1

Title	Acronym	Source	Rank
Empirical Methods in Natural Language Processing	EMNLP	CORE2023	A*

Corporate R&D bodies provide incentives to employees if they have papers accepted at such venues, as that often boosts the global reputation of the corporate body. Implement a scheme to decide on the Incentive given the Conference Rank using a) if-else-if, b) switch as below:

Conference Rank	Incentive
A*	500 USD + funds for conference travel, accommodation, and registration
A	300 USD + funds for conference travel and accommodation
B	150 USD + funds for conference travel
C	100 USD only

Can you also modify your switch scheme (hint: adjusting 'break') if the company has the same incentive for both A* and A?

Also, note that "A*" can not be expressed as a character. So, you need to have a number or character mapping of the conference ranks.

Set 4

1. Write programs to print the following patterns (for any n) using loops:

(a) For n = 10, output:

```
10
10 9
10 9 8
....
10 9 8 7 6 5 4 3 2 1
```

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

```
void main()
{
    int i, j, n = 10;

    for(i = 1 ; i <= n ; i++)
    {
        for(j = 1 ; j <= i ; j++)
        {
            printf("%d ", n - j + 1);
        }
        printf("\n");
    }

}
```

(b) For n = 5, output:

```
1
1 1
1 1 2
1 1 2 3
1 1 2 3 5
```

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

```
void main()
```

```
{
```

```
    int i, j, n = 10, prev_1, prev_2, current;//prev_1: first number, prev_2: number  
following prev_1
```

```
    for(i = 1 ; i <= n ; i++)
```

```
    {
```

```
        prev_1 = 1;
```

```
        prev_2 = 1;
```

```
        for(j = 1 ; j <= i ; j++)
```

```
        {
```

```
            if (j <= 2)
```

```
            {
```

```
                printf("1 ");//Ignore for the first two numbers
```

```
            }
```

```
            else
```

```
            {
```

```
                current = prev_1 + prev_2;
```

```
                printf("%d ", current);
```

```
                prev_1 = prev_2;
```

```
                prev_2 = current;//(prev_1, prev_2) window shifts one
```

```
place to the right
```

```
            }
```

```
        }
```

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

```
    }
```

```
}
```

(c) For n = 5, output:

```
    1  
  1 1  
 1 2 1  
1 3 3 1  
1 4 6 4 1
```

```

#include<stdio.h>

void main()
{
    int i, j, n = 5, t, arr[10][10];
    for (i = 0; i < n; i++)
    {
        for(t = 1; t <= 10 - i ; t++)
        {
            printf(" "); //printing leading blank spaces
        }
        for (j = 0; j <= i; j++)
        {
            // First and last values in every row are 1
            if (j == i || j == 0)
            {
                arr[i][j] = 1;
            }

            // Other values are sum of values just above and left of above
            else
            {
                arr[i][j] = arr[i - 1][j - 1] + arr[i - 1][j];
            }

            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}

```

(d) For n = 5, output:

```

5
5 4 5
5 4 3 4 5
5 4 3 2 3 4 5
5 4 3 2 1 2 3 4 5

```



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

```
void main()
{
    int i, j, k, n = 5;

    for(i = n ; i >= 1; i--)
    {
        for(j = n ; j >= n - (n - i); j--)
        {
            printf("%d ", j);

        }
        for(k = j + 2 ; k <= n; k++)
        {
            printf("%d ", k);

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

(e) For n = 4, output

```
1 2 3 4
2 3 4
3 4
4
```

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

```
void main()
{
    int i, j, k, n = 4;

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

```

        printf("%d ", j);
    }
    printf("\n");
}
}

```

2. Implement a login interface using a do-while loop, asking a person his/her username and password, and allowing access in case of successful authentication. Assume that you have a known list of usernames in an array: UserNames, and the corresponding passwords (in the same order) in another array: Passwords. That is, array element UserName[i] corresponds to array element Passwords[i].

Once the person inputs the username, *search* for the same in UserNames, and if it matches an element in UserNames, ask for the password, which you search in Passwords. You may use the code of “linear search” on slide number 23 of “Loops” on the WeLearn page for the course.

If the username and password match, print a message ‘Welcome!’ In case of a mismatch, print an appropriate message and ask if the person wants to continue; if yes, ask for the username and password again; else, terminate the loop.

Note: You may assume that both the username and passwords are integers. In the later versions, you should be able to consider strings.

Considering strings, the solution:

```

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

int main()
{
    char uname[10], *usernames[] = {"kripa", "ayan", "doi"};
    char pswrd[10], *passwords[] = {"kripa123", "gupibagha", "sukti"};
    int flag = -1, choice, i;

    do
    {
        flag = -1;
        printf("Give your username: ");
        scanf("%s", uname);
        for(i = 0; i < 3 ; i++)

```

```

    {
        if(!strcmp(uname, usernames[i]))
        {
            flag = i;
            break;
        }
    }
    if(flag > -1)
    {
        printf("Give your password: ");
        scanf("%s", pswrd);

        if(!strcmp(pswrd, passwords[flag]))
        {
            printf("Welcome!\n");
            break;
        }
        else{
            printf("Invalid credentials!\n");
        }
    }
    else{
        printf("Invalid credentials!\n");
    }

    printf("Do you want to continue? (1/0)?");
    scanf("%d", &choice);
}
while(choice);
}

```

3. Modify the code of Q8 of Set 3 to incorporate a message (using an appropriate loop) asking if the user wants to continue. If the user does, s/he should be shown the catalog again with updated stock (if the user did choose to buy some items in the first pass, the stock should show reduced quantities of the corresponding items) and update the total quantity of items (for 2 bananas, 3 mangoes this number should be 5) bought with the total bill to be paid. The user

interface should continue asking if the user wishes to buy anything else until the user wishes not to continue. The total bill will be shown only when the user chooses not to continue.

4. Design an employee leave management system. An employee can have Casual Leave, Special Casual Leave, Vacation Leave, On-duty Leave, and Restricted Holiday Leave, with details shown in the table below. The “Nos” are the number of leaves for each employee.

SI no.	Leave Type	Nos	Prerequisites
1	Casual Leave	30	None
2	Special Casual leave	15	Invitation Letter
3	On-duty Leave	10	Invitation Letter
4	Vacation Leave	60	Should be only for June, July, and December
5	Restricted Holiday Leave	5	Should appear in the list of restricted holidays

Assume that there are 10 employees with unique employee IDs. An employee applying for leave should be able to log in using their username and password (you may reuse the solution from Q2). Once the login is successful, you should ask the employee to choose from the list of leave types showing the number of leave available for that employee. The employee, after choosing the type of leave, also enters the number of days of leave applied (except restricted holiday leave, where only one day at a time is permissible). If that is available, grant immediately for type 1 (Casual Leave) and update the Nos.

For types 2 and 3, ask the employee if the “invitation letter” is available; if yes, grant and update; otherwise, deny.

For Vacation leave, also ask for the month for which the leave is applied, and grant based on availability if the month is June, July, or December.

For restricted holidays, maintain a record of restricted holidays (day, month) in array(s), storing only for the current year. Ask the employee for the day and month for which leave is requested (restricted holidays can be granted only one day at a time). If the requested date appears in the permitted days (through a search in the array(s)), grant; otherwise, deny.

For all the leave types, the leave nos will be updated if granted, and a request for a number exceeding the available nos will be denied. The Prerequisites for leave also need to be adhered with. The balance leave will be shown again, and the employee will be asked if the employee wants to continue. Act according to the choice provided using an appropriate loop. After the loop ends, print the final leave balance for that employee for all the leave types.

Note: This is expected to be a bare-minimum version (call it v1). You should be able to upgrade to superior versions later, after topics like string handling (to handle non-integer variables), I/O (reading and writing floating points, strings, formatting, etc.), functions (modularization; building your own header files), structures (encapsulation of all the details of one employee), files (creation of permanent employee database on hard drive, loading, updating them), complexity of algorithms (writing efficient implementations), efficient searching in the employee database (e.g. binary search) etc. are covered.

Complexity of algorithms:

5. Consider an array A = [2, 4, 6, 8, 10, 12, 14, 0, 0, 0, 0], where 0s specify empty slots. Note that the non-zero subarray in the array is sorted. Insert a new element x (input from the user) in array A at the end (replacing the first 0) by searching for the first 0. Now implement a search algorithm to search for any element in the array. Compute the time complexity (using Big O) and put it as a comment in your code.

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

```
void insertElement(int* A, int size, int x) {  
    // Search for the first zero  
    for (int i = 0; i < size; i++) {  
        if (A[i] == 0) {  
            A[i] = x; // Insert the element  
            printf("Inserted %d at position %d\n", x, i);  
            return;  
        }  
    }  
    printf("No empty slots available!\n");  
}
```

```
// Time Complexity: O(n) - Linear search through the array
```

```
int searchElement(int *A, int size, int key) {  
    for (int i = 0; i < size; i++) {  
        if (A[i] == key) {  
            return i; // Return index if found  
        }  
    }  
    return -1; // Return -1 if not found
```

```
}
```

```
// Function to print the array
void printArray(int* A, int size) {
    printf("Array: [");
    for (int i = 0; i < size; i++) {
        printf("%d", A[i]);
        if (i < size - 1) {
            printf(", ");
        }
    }
    printf("]\n");
}
```

```
int main() {
    int A[] = {2, 4, 6, 8, 10, 12, 14, 0, 0, 0, 0};
    int size = sizeof(A) / sizeof(A[0]);
    int x, search_target, result;

    // Get input from user
    printf("Enter element to insert: ");
    scanf("%d", &x);

    printf("Before Insertion:\n");
    printArray(A, size);

    // Insert the element
    insertElement(A, size, x);

    printf("After Insertion:\n");
    printArray(A, size);

    // Search for an element
    printf("Enter element to search: ");
    scanf("%d", &search_target);

    result = searchElement(A, size, search_target);

    if (result != -1) {
        printf("Element %d found at index %d\n", search_target, result);
    } else {
        printf("Element %d not found in the array\n", search_target);
    }
}
```

```
    return 0;
}
```

6. Improve the insertion method in Q5 by inserting at an appropriate location in the array such that the non-zero elements in it remain sorted. Now perform the search using a better search algorithm (better than $O(n)$) in terms of Big O.

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

```
// Function to find the number of non-zero elements
```

```
int findNonZeroCount(int* A, int size) {
    int count = 0;
    for (int i = 0; i < size && A[i] != 0; i++) {
        count++;
    }
    return count;
}
```

```
// Function to insert element at appropriate position to maintain sorted order
```

```
void insertElementSorted(int* A, int size, int x) {
    int nonZeroCount = findNonZeroCount(A, size);
```

```
    // Check if array is full
```

```
    if (nonZeroCount >= size) {
        printf("Array is full! Cannot insert %d\n", x);
        return;
    }
```

```
    // Find the correct position to insert
```

```
    int pos = 0;
    while (pos < nonZeroCount && A[pos] < x) {
        pos++;
    }
```

```
    // Shift elements to the right to make space
```

```
    for (int i = nonZeroCount; i > pos; i--) {
        A[i] = A[i - 1];
    }
```

```
    // Insert the new element
```

```
    A[pos] = x;
    printf("Inserted %d at position %d\n", x, pos);
}
```

// Time Complexity: $O(\log n)$ - Binary search on sorted portion

```
int binarySearch(int *A, int size, int target) {
```

```
    int nonZeroCount = findNonZeroCount(A, size);
```

```
    int left = 0;
```

```
    int right = nonZeroCount - 1;
```

```
    while (left <= right) {
```

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

```
        if (A[mid] == target) {
```

```
            return mid; // Element found
```

```
        }
```

```
        if (A[mid] < target) {
```

```
            left = mid + 1;
```

```
        } else {
```

```
            right = mid - 1;
```

```
        }
```

```
    }
```

```
    return -1; // Element not found
```

```
}
```

// Function to print the array

```
void printArray(int *A, int size) {
```

```
    printf("Array: [");
```

```
    for (int i = 0; i < size; i++) {
```

```
        printf("%d", A[i]);
```

```
        if (i < size - 1) {
```

```
            printf(", ");
```

```
        }
```

```
    }
```

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

```
}
```

```
int main() {
```

```
    int A[] = {2, 4, 6, 8, 10, 12, 14, 0, 0, 0, 0};
```

```
    int size = sizeof(A) / sizeof(A[0]);
```

```
    int x, search_target, result;
```

```
    printf("Initial ");
```

```
    printArray(A, size);
```



```

printf("\n");

// Get input from user
printf("Enter element to insert: ");
scanf("%d", &x);

// Insert the element while maintaining sorted order
insertElementSorted(A, size, x);

printf("\nAfter insertion ");
printArray(A, size);
printf("\n");

// Search for an element using binary search
printf("Enter element to search: ");
scanf("%d", &search_target);

result = binarySearch(A, size, search_target);

if (result != -1) {
    printf("Element %d found at index %d\n", search_target, result);
} else {
    printf("Element %d not found in the array\n", search_target);
}

return 0;
}

```

7. Assume you have a database of user names and passwords (integers, stored in arrays). Implement a system that allows users to create new user accounts (username and password) and expand the database of user accounts. Later, the same database is searched for usernames and passwords. Design a scheme that makes this search efficient in terms of Big O.

Strings:

8. Write a C program to take the first name (e.g., "Sachin") and surname (e.g., "Tendulkar") as input in separate strings and using string operations to store the full name ("Sachin Tendulkar") as a single string and print it.

```

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

```

```

void main()
{
    char fname[20], surname[20], name[40];

    printf("Give first name:");
    scanf("%s", fname);
    printf("Give surname: ");
    scanf("%s", surname);

    strcpy(name, fname);
    strcat(name, " ");
    strcat(name, surname);
    printf("\nFull name: %s", name);

}

```

9. Write a C program to take first name (e.g., "Sachin"), middle name (e.g., "Ramesh") and surname (e.g., "Tendulkar") as input as separate strings, use string operations to store the full name in the format: first_name first_letter_of_middlename. surname (e.g., "Sachin R. Tendulkar") in another string and print it.

```

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

void main()
{
    char fname[20], surname[20], midname[20], name[40], midname_short[3];

    printf("Give first name:");
    scanf("%s", fname);
    printf("Give middle name:");
    scanf("%s", midname);
    printf("Give surname: ");
    scanf("%s", surname);
    midname_short[0] = midname[0];
    midname_short[1] = '.';
    midname_short[2] = '\0';

    strcpy(name, fname);
    strcat(name, " ");
    strcat(name, midname_short);

```

```

        strcat(name, " ");
        strcat(name, surname);
        printf("\nFull name: %s", name);

    }

```

10. Implement Q2 considering username and password as strings and with search having $O(\log n)$ complexity.

Set 5

1. Implement a function MAXINT2() that takes two integers as arguments and returns the maximum of them. Take two integers as input, and using MAXINT2(), print the maximum. Similarly, implement MININT2() and print the minimum of the same integers.
2. Take three integers as input and, using MAXINT2() print the maximum of these three integers. Using MININT2(), similarly, print the minimum of the same integers.
3. Write two functions: i) INCVAL() – that takes the value of an integer as an argument and returns its incremented value, and ii) INCREF() – that takes the address of an integer as an argument and increments its value. Call these functions from main() to increment two local variables in main() – one using the first function and the other using the second one, and then print the incremented values of these two local variables in main().

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

```
// Takes integer value as argument and returns incremented value
```

```
int INCVAL(int num) {
    return num + 1;
}
```

```
// Takes address of integer as argument and increments its value
```

```
void INCREF(int *num_ptr) {
    (*num_ptr)++;
}
```

```
int main() {
```

```
    int var1 = 5; // Variable for INCVAL
    int var2 = 10; // Variable for INCREF
}
```

```

printf("Initial values:\n");
printf("var1 = %d, var2 = %d\n\n", var1, var2);

// Increment var1 using INCVAL (pass by value)
var1 = INCVAL(var1);
printf("After INCVAL(var1):\n");
printf("var1 = %d, var2 = %d\n\n", var1, var2);

// Increment var2 using INCREF (pass by reference)
INCREF(&var2);
printf("After INCREF(&var2):\n");
printf("var1 = %d, var2 = %d\n\n", var1, var2);

return 0;
}

```

4. Consider an integer array named **marks** containing the marks (out of 100) of five students. Define a function `grace_marks()`, which takes an array of integers, along with the number of elements in the array, as arguments, and adds grace marks of 5 to those students who obtained less than 40. Print the values of the array **marks** before and after passing the array to `grace_marks()`.

```

void grace_marks(int *marks, int n)
{
    int i;

    for(i = 0; i < n; i++)
    {
        if(marks[i] < 40)
        {
            marks[i] += 5;
        }
    }
}

```

5. Consider an integer array **A** of integers and a function `AVGINTARRAY()` that takes **A** (with the number of elements in it) as arguments, returning the average of the elements in **A**. Note that although the elements in **A** are integers, their average can be a real number; so please define `AVGINTARRAY()` accordingly.

```

float AVGINTARRAY(int *a, int n)
{
    int i, sum = 0;

    for(i = 0; i < n; i++)
    {
        sum += a[i];
    }

    return ((1.0)*sum)/n;
}

```

6. Create a header file “distance.h” where you define three distance measures: Euclidean, Manhattan, and Chebyshev distance as functions and their bodies in “distance.c”; finally calling them from main after including your header. The details of the distance measures are as below.

In **n-dimensional real space**, the distance between two points $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ can be defined in multiple ways; some examples are as follows.

- **Euclidean distance** or “straight-line distance” (L_2 norm) is perhaps the most used measure in science and machine learning, defined as:

$$d_2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$




- **Manhattan distance**, taxi-cab distance, or city block distance (L_1 norm) also has applications across fields, defined as:

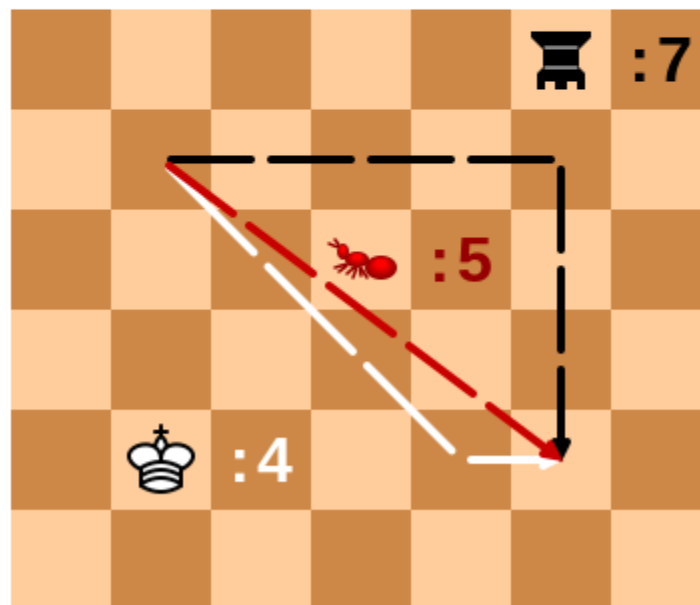
$$d_1(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- **Chebyshev distance** is defined as below:

$$d_\infty(x, y) = \max_i |x_i - y_i|$$

Visually, these distances for the hypotenuse of a 3-4-5 triangle on a chessboard are as shown below:

1	1	1	$\sqrt{2}$	1	$\sqrt{2}$	2	1	2
1		1	1		1	1		1
1	1	1	$\sqrt{2}$	1	$\sqrt{2}$	2	1	2
Chebyshev			Euclidean			Taxicab		



In a program, include "distance.h", pass two points x and y, each of 5 dimensions (that is, points of 5-dimensional Real space), call the three functions: Euclidean(), Manhattan(), and Chebyshev() passing x and y, finally print the three distance values. Note that x, y will be arrays of real type (e.g., float), fabs() and sqrt() in math.h will be needed for finding the absolute value and square root, respectively.

distance.h

=====

// Function to compute L1 (Manhattan) norm

double L1_norm(double *x, double *y, int n);

// Function to compute L2 (Euclidean) norm

double L2_norm(double *x, double *y, int n);

// Function to compute Chebyshev (L^∞) norm

double Chebyshev_norm(double *x, double *y, int n);

distance.c

=====

#include <stdio.h>

#include <math.h>

// Function to compute L1 (Manhattan) norm

double L1_norm(double *x, double *y, int n) {

double sum = 0.0;

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

sum += fabs(x[i] - y[i]);

return sum;

}

// Function to compute L2 (Euclidean) norm

double L2_norm(double *x, double *y, int n) {

double sum = 0.0;

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

sum += pow(x[i] - y[i], 2);

return sqrt(sum);

}

// Function to compute Chebyshev (L^∞) norm

double Chebyshev_norm(double *x, double *y, int n) {

double max = 0.0;

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

double diff = fabs(x[i] - y[i]);

if (diff > max)

max = diff;

}

return max;

```
}
```

```
function_norm-distances.c
```

```
=====
```

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

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

```
#include "distance.h"
```

```
int main() {
```

```
    double x[] = {2.0, 3.0, 4.0, 5.0};
```

```
    double y[] = {1.0, 1.0, 2.0, 3.0};
```

```
    int n = sizeof(x) / sizeof(x[0]);
```

```
    printf("L1 (Manhattan) distance: %.3f\n", L1_norm(x, y, n));
```

```
    printf("L2 (Euclidean) distance: %.3f\n", L2_norm(x, y, n));
```

```
    printf("Chebyshev ( $L^\infty$ ) distance: %.3f\n", Chebyshev_norm(x, y, n));
```

```
    return 0;
```

```
}
```

Compilation:

```
gcc -o function_norm-distances function_norm-distances.c distance.c -lm
```

Execution:

```
./function_norm-distances
```

7. Take three integers a, b and c as input. Store their addresses in pointers p, q and r, respectively. Using these pointers, print the sum and product of a, b and c.

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

```
void main()
```

```
{
```

```
    int a = 1, b = 2, c = 6;
```

```
    int *p, *q, *r;
```

```
    p = &a;
```

```
    q = &b;
```

```
    r = &c;
```



```

    printf("sum=%d prod=%d\n", *p + *q + *r, *p**q**r);
}

```

8. Write a C program to take any string (e.g. "Programming in C language") and define two functions: i) count(): that counts the number of occurrences of a character in the string (e.g. it returns 3 for 'a' in the example string) and ii) replace(): that replaces a character in the input string with another character (e.g. for the input string if 'a' is to be replaced with 'z', the resulting output string should be "Progrzmming in C lznguzge"). You should pass the string as a pointer to these two functions. Use of a global variable is NOT allowed.

```

#include<stdio.h>
#include<string.h>
int count(char *str, char ch);
void replace(char *str, char ch1, char ch2);
int main(void)
{
    char str[30];
    strcpy(str,"Programming in C language");
    printf("%d\n",count(str,'a'));
    replace(str,'a','z');
    puts(str);
    return 0;
}
int count(char *str, char ch)
{
    int cnt=0;
    while(*str++!='\0')
        if(*str==ch)
            cnt++;
    return cnt;
}
void replace(char *str, char ch1, char ch2)
{
    while(*str++!='\0')
        if(*str==ch1)
            *str=ch2;
}

```

9. Assume that all the secret codewords (integers) of the members of a gang "Prime Suspects" have a property: the sum of the digits is a prime number (e.g., code word: 111, 10523 etc.). Write a C program to implement the following scheme. A person inputs the codeword (taken as input by a function getCodeword() and returned to main()). This inputted codeword passes

through functions: sumOfDigits() and isPrime() to check whether the aforementioned property is satisfied by the codeword. If yes, there should be a welcome message like “Welcome to Prime Suspect gang!”; else, print an intruder alert message.

```
#include <stdio.h>

int sumOfDigits(int n)
{
    int sum = 0, num = n;

    while(num)
    {
        sum += num%10;
        num = num/10;
    }

    return sum;
}

int isPrime(int nn)
{
    int i, p = 1;

    for(i = 2; i<=nn/2; i++)
    {
        if(nn%i == 0)
        {
            p = 0;
            break;
        }
    }

    return p;
}

int getCodeword()
{
    int codeword;
    printf("Give your code word: ");
    scanf("%d", &codeword);
    return codeword;
}

int main() {
```

```

    int code = getCodeword();

    if(isPrime(sumOfDigits(code)))
    {
        printf("Welcome to Prime Suspect gang!\n");
    }
    else
    {
        printf("Intruder alert!!");
    }

    return 0;
}

```

10. Assume that $\sin(x)$ can be expressed as the sum of the series: $x - x^3/3! + x^5/5! - \dots$; for any x in radians. Implement a function $\text{Sin}(x, n)$ in C, where n is the number of terms to be considered in the sum. In addition, implement a factorial function. You can use the $\text{pow}()$ function in math.h and $\sin(x)$ in the same header file to compare the computed value by $\text{Sin}()$.

```

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

int fact(int n)
{
    int i, prod = 1;

    for(i = 1; i <= n; i++)
    {
        prod *= i;
    }

    return prod;
}

float Sin(float x, int n)
{
    float sum = 0.0;
    int i;

    for(i = 0; i <= n; i++)
    {
        sum += (pow(-1, i)/(float)fact(2*i+1))*pow(x, 2*i + 1);
    }
}

```

```

        return sum;
    }

    int main()
    {

        float x = 1.5707;
        printf("Series sum: %f\n", Sin(x, 10));
        printf("Actual value: %f\n", sin(x));

        return 0;
    }

```

11. Implement a function F() that takes a first name (length not more than 9 characters) as an argument and “returns” the encrypted key formed by the first character, the last character, and appending the length of the name. E.g., if the first name is “James” the key is “Js5”. Note that to convert a single-digit integer (say, 5) to the corresponding character (here '5') in C, you can add the ASCII value of the character '0' to the integer. Now, you should implement F() in two different ways: i) you pass the ONLY name as an argument and return the key, and ii) you pass BOTH the name and a variable that will store the key as arguments but returns nothing.

```

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

char* encrypt(char *s)
{
    //char s_encoded[20];
    char *s_encrypted;

    s_encrypted = (char*)malloc(20*sizeof(char));

    s_encrypted[0] = s[0];
    s_encrypted[1] = s[strlen(s)-1];
    s_encrypted[2] = strlen(s) + '0';
    s_encrypted[3] = '\0';

    return s_encrypted;
}

void encrypt_noreturn(char *s, char *s_encrypted)

```

```

{

    s_encrypted[0] = s[0];
    s_encrypted[1] = s[strlen(s)-1];
    s_encrypted[2] = strlen(s) + '0';
    s_encrypted[3] = '\0';

}

int main()
{
    char s[] = "James", *s_encrypted, s_e[10];

    //s_encrypted = encrypt(s);

    //printf("%s is encrypted as %s\n", s, s_encrypted);

    encrypt_noreturn(s, s_e);

    printf("%s is encoded as %s\n", s, s_e);

}

```

12. Implement a student database with 5 students, each having name, roll no, and CGPA so far stored using **struct**. Take a roll number as input from the user and print the corresponding CGPA. You may assume that the student database exists in an array (in main). You should pass the array to a function for searching the student, and if found, return the CGPA.