

## DAY 14

1. #include<stdio.h>

struct test

{

/\*char a;

int b;

short c;

char d;

int e;\*/

int b;

int e;

short c;

char a;

char d;

};

int main()

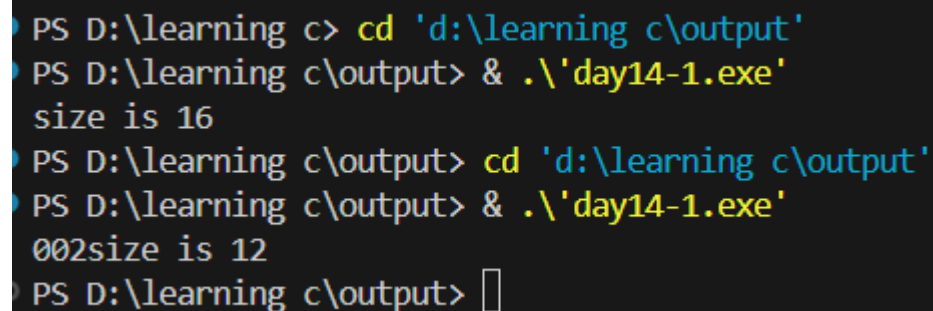
{

struct test eg;

//printf("001size is %d \n",sizeof(eg));//output=16

printf("002size is %d \n",sizeof(eg));//output=12

}



```
PS D:\learning c> cd 'd:\learning c\output'
PS D:\learning c\output> & .\day14-1.exe
size is 16
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\day14-1.exe
002size is 12
PS D:\learning c\output> █
```

2. //accessing structure member through pointer using dynamic memory allocation

```

#include<stdio.h>

#include<stdlib.h>

struct course
{
    int marks;
    char subject[30];

};

int main()
{
    //i want to use the same structure for multiple courses

    struct course *cpointer;

    int n;

    printf("enter the no.of records \n");
    scanf("%d",&n);

    //dynamicmemry allocation for no.of records
    cpointer=(struct course *)malloc(n*sizeof(struct course));

    if(0==cpointer)
    {
        printf("not alllocated \n");
    }
    else{
        printf("memory allocated \n");
    }

    for(int i=0;i<n;i++)
    {
        printf("enter subject nameand marks \n");
        scanf("%s %d", (cpointer+i)->subject, &(cpointer+i)->marks);
    }

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

```

```

{
    printf("%s %d \n", (cpointer+i)->subject, (cpointer+i)->marks);
}

free(cpointer);

return 0;
}

```

```

C/C++ Compile Run - output
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-2.exe'
enter the no.of records
4
memory allocated
enter subject nameand marks
eng 90
enter subject nameand marks
hindi 80
enter subject nameand marks
chem 89
enter subject nameand marks
phy 78
eng 90
hindi 80
chem 89
phy 78
PS D:\learning c\output>

```

### 3. Problem Statement: Employee Records Management

Write a C program to manage a list of employees using **dynamic memory allocation**. The program should:

1. Define a structure named Employee with the following fields:
  - id (integer): A unique identifier for the employee.
  - name (character array of size 50): The employee's name.
  - salary (float): The employee's salary.
2. Dynamically allocate memory for storing information about n employees (where n is input by the user).

3. Implement the following features:

- **Input Details:** Allow the user to input the details of each employee (ID, name, and salary).
- **Display Details:** Display the details of all employees.
- **Search by ID:** Allow the user to search for an employee by their ID and display their details.
- **Free Memory:** Ensure that all dynamically allocated memory is freed at the end of the program.

**Constraints**

- $n$  (number of employees) must be a positive integer.
- Employee IDs are unique.

**Sample Input/Output**

**Input:**

*Enter the number of employees: 3*

*Enter details of employee 1:*

*ID: 101*

*Name: Alice*

*Salary: 50000*

*Enter details of employee 2:*

*ID: 102*

*Name: Bob*

*Salary: 60000*

*Enter details of employee 3:*

*ID: 103*

*Name: Charlie*

*Salary: 55000*

*Enter ID to search for: 102*

**Output:**

*Employee Details:*

*ID: 101, Name: Alice, Salary: 50000.00*

*ID: 102, Name: Bob, Salary: 60000.00*

*ID: 103, Name: Charlie, Salary: 55000.00*

*Search Result:*

*ID: 102, Name: Bob, Salary: 60000.00*

```
        free(emp);

        return 0;
    }
#include <stdio.h>
#include <stdlib.h>

struct Employee {
    int id;
    char name[50];
    float salary;
};

int main() {
    struct Employee *emp;
    int n;

    printf("Enter the number of employees: \n");
    scanf("%d", &n);

    emp = (struct Employee *)malloc(n * sizeof(struct Employee));
```

```
if (emp == NULL) {  
    printf("Memory not allocated.\n");  
    return 1; // Exit the program if memory allocation fails  
} else {  
    printf("Memory allocated.\n");  
}
```

```
printf("Input details: \n");  
for (int i = 0; i < n; i++) {  
    printf("Enter details of employee %d:\n", i + 1);  
    printf("ID: ");  
    scanf("%d", &emp[i].id);  
    printf("Name: ");  
    scanf("%s", emp[i].name);  
    printf("Salary: ");  
    scanf("%f", &emp[i].salary);  
}
```

```
printf("\nEmployee details: \n");  
for (int i = 0; i < n; i++) {  
    printf("ID: %d, Name: %s, Salary: %.2f\n", emp[i].id, emp[i].name, emp[i].salary);  
}  
printf("enter an id \n");  
int id;  
scanf("%d",&id);  
int found=0;
```

```
for(int i=0;i<n;i++)  
{  
    if(id==emp[i].id)  
    {
```

```
    printf("ID: %d, Name: %s, Salary: %.2f\n", emp[i].id, emp[i].name, emp[i].salary);  
    found=1;  
    break;  
}  
  
}  
if(!found)  
{  
    printf("not found \n");  
}  
  
free(emp);  
  
return 0;  
}
```

```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-3.exe'
Enter the number of employees:
2
Memory allocated.
Input details:
Enter details of employee 1:
ID: 1
Name: rinta
Salary: 4000
Enter details of employee 2:
ID: 2
Name: riya
Salary: 50000

Employee details:
ID: 1, Name: rinta, Salary: 4000.00
ID: 2, Name: riya, Salary: 50000.00
enter an id
2
ID: 2, Name: riya, Salary: 50000.00
PS D:\learning c\output> 

```

#### 4.Problem 1: Book Inventory System

##### Problem Statement:

Write a C program to manage a book inventory system using dynamic memory allocation. The program should:

1. Define a structure named Book with the following fields:
  - id (integer): The book's unique identifier.
  - title (character array of size 100): The book's title.
  - price (float): The price of the book.
2. Dynamically allocate memory for n books (where n is input by the user).
3. Implement the following features:
  - **Input Details:** Input details for each book (ID, title, and price).
  - **Display Details:** Display the details of all books.
  - **Find Cheapest Book:** Identify and display the details of the cheapest book.
  - **Update Price:** Allow the user to update the price of a specific book by entering its ID.

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



```
#include<stdlib.h>

struct Book
{
    int id;
    char title[100];
    float price;
};

int main()
{
    int n;

    struct Book *book;

    printf("enter the number of books \n");
    scanf("%d",&n);

    book= (struct Book *)malloc(n * sizeof(struct Book));

    if (book == NULL) {
        printf("Memory not allocated.\n");
        // Exit the program if memory allocation fails
    } else {
        printf("Memory allocated.\n");
    }

    //int bid=0;

    printf("Input details: \n");

    for (int i = 0; i < n; i++) {
        printf("Enter details of book %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &book[i].id);
        printf("title: ");
        scanf("%s", book[i].title);
        printf("price: ");
        scanf("%f", &book[i].price);
    }
}
```

```

printf("book details \n");
for (int i = 0; i < n; i++) {
    printf("ID: %d, Title: %s, Price: %.2f\n", book[i].id, book[i].title, book[i].price);
}

float cheap= book[0].price;

for(int i=0;i<n;i++)
{
    if(book[i].price<=cheap)
    {
        printf("cheapest book is %d %s %.2f\n",book[i].id,book[i].title,book[i].price);
        break;
    }
    else
    {
        printf("cheapest book is %d %s %.2f\n",book[0].id,book[0].title,book[0].price);

    }
}

printf("enter an id \n");
int id;
scanf("%d",&id);
int found=0;
float price;

for(int i=0;i<n;i++)
{
    if(id==book[i].id)
    {
        printf("update price \n");
        scanf("%f",&price);
    }
}

```

```
book[i].price=price;
```

```
found=1;
```

```
printf("ID: %d, Title: %s, Price: %.2f\n", book[i].id, book[i].title, book[i].price);
```

```
break;
```

```
}
```

```
}
```

```
if(!found)
```

```
{
```

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

```
}
```

```
free(book);
```

```
}
```

```
PS D:\learning c> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-4.exe'
enter the number of books
2
Memory allocated.
Input details:
Enter details of book 1:
ID: 1
title: snowwhite
price: 560
Enter details of book 2:
ID: 2
title: heirs
price: 890
book details
ID: 1, Title: snowwhite, Price: 560.00
ID: 2, Title: heirs, Price: 890.00
cheapest book is 1 snowwhite 560.00
enter an id
2
update price
900
ID: 2, Title: heirs, Price: 900.00
PS D:\learning c\output> █
```

## 5.Problem 2: Dynamic Point Array

### Problem Statement:

Write a C program to handle a dynamic array of points in a 2D space using dynamic memory allocation. The program should:

1. Define a structure named Point with the following fields:
  - x (float): The x-coordinate of the point.
  - y (float): The y-coordinate of the point.
2. Dynamically allocate memory for n points (where n is input by the user).
3. Implement the following features:
  - **Input Details:** Input the coordinates of each point.
  - **Display Points:** Display the coordinates of all points.

- **Find Distance:** Calculate the Euclidean distance between two points chosen by the user (by their indices in the array).
- **Find Closest Pair:** Identify and display the pair of points that are closest to each other.

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

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

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

```
#include <float.h>
```

```
struct Point {
```

```
    float x;
```

```
    float y;
```

```
};
```

```
int main() {
```

```
    struct Point *p;
```

```
    int n;
```

```
    printf("Enter the max number of points:\n");
```

```
    scanf("%d", &n);
```

```
    p = (struct Point *)malloc(n * sizeof(struct Point));
```

```
    if (p == NULL) {
```

```
        printf("Memory not allocated.\n");
```

```
        return 0;
```

```
    } else {
```

```
        printf("Memory allocated.\n");
```

```
    }
```

```
    printf("Input details:\n");
```

```
for (int i = 0; i < n; i++) {  
    printf("Enter X for point %d: ", i + 1);  
    scanf("%f", &p[i].x);  
    printf("Enter Y for point %d: ", i + 1);  
    scanf("%f", &p[i].y);  
}
```

```
printf("Points are:\n");  
for (int i = 0; i < n; i++) {  
    printf("Point %d: X = %.2f, Y = %.2f\n", i + 1, p[i].x, p[i].y);  
}
```

```
int index1, index2;  
  
printf("Enter the indices of the two points to calculate the distance (1 to %d):\n",  
n);  
  
scanf("%d %d", &index1, &index2);  
  
index1--;  
index2--;
```

```
if (index1 < 0 || index1 >= n || index2 < 0 || index2 >= n) {  
    printf("Invalid indices. Please choose indices between 1 and %d.\n", n);  
} else {  
    float euclid = sqrt(pow(p[index2].x - p[index1].x, 2) + pow(p[index2].y -  
p[index1].y, 2));  
    printf("The Euclidean distance between point %d and point %d is %.2f\n",  
index1 + 1, index2 + 1, euclid);  
}
```

```
float min_distance = FLT_MAX;  
  
int closest_index1 = 0, closest_index2 = 1;
```

```

for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        float distance = sqrt(pow(p[j].x - p[i].x, 2) + pow(p[j].y - p[i].y, 2));
        if (distance < min_distance) {
            min_distance = distance;
            closest_index1 = i;
            closest_index2 = j;
        }
    }
}

printf("The closest pair of points are point %d and point %d with a distance of\n", closest_index1 + 1, closest_index2 + 1, min_distance);

free(p);

return 0;
}

```

```

PS D:\learning c\output> & .\'day14-5.exe'
Enter the max number of points:
3
Memory allocated.
Input details:
Enter X for point 1: 1
Enter Y for point 1: 2
Enter X for point 3: 5
Enter Y for point 3: 6
Points are:
Point 1: X = 1.00, Y = 2.00
Point 2: X = 3.00, Y = 4.00
Point 3: X = 5.00, Y = 6.00
Enter the indices of the two points to calculate the distance (1 to 3):
1 2
The Euclidean distance between point 1 and point 2 is 2.83
The closest pair of points are point 1 and point 2 with a distance of 2.83
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-5.exe'
Enter the max number of points:
3
Memory allocated.
Input details:
Enter X for point 1: 1
Enter Y for point 1: 2
Enter X for point 2: 3
Enter Y for point 2: 4
Enter X for point 3: 5
Enter Y for point 3: 6
Points are:
Point 1: X = 1.00, Y = 2.00
Point 2: X = 3.00, Y = 4.00
Point 3: X = 5.00, Y = 6.00
Enter the indices of the two points to calculate the distance (1 to 3):
1 3
The Euclidean distance between point 1 and point 3 is 5.66
The closest pair of points are point 1 and point 2 with a distance of 2.83
PS D:\learning c\output> 

```

6. //union

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

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

```
struct course
```

```
{
```

```
    int marks;
```

```
    char subject[30];
```



```

};

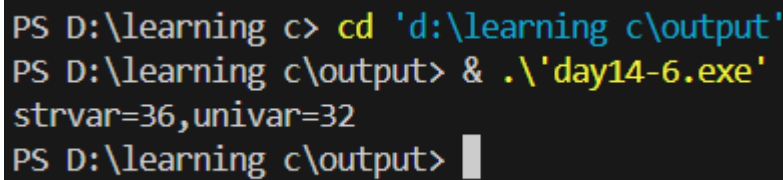
union course1
{
    int marks;
    //char s;
    char subject[30];

};

int main()
{

    struct course cpointer;
    union course1 u1;
    printf("strvar=%d,univar=%d \n",sizeof(cpointer),sizeof(u1));
}

```



```

PS D:\learning c> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-6.exe'
strvar=36,univar=32
PS D:\learning c\output>

```

7. //union

```

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

```

```

union course1
{
    int a;
    int b;
}

```

```
}u1;
```

```
int main()
```

```
{
```

```
    u1.a=10;
```

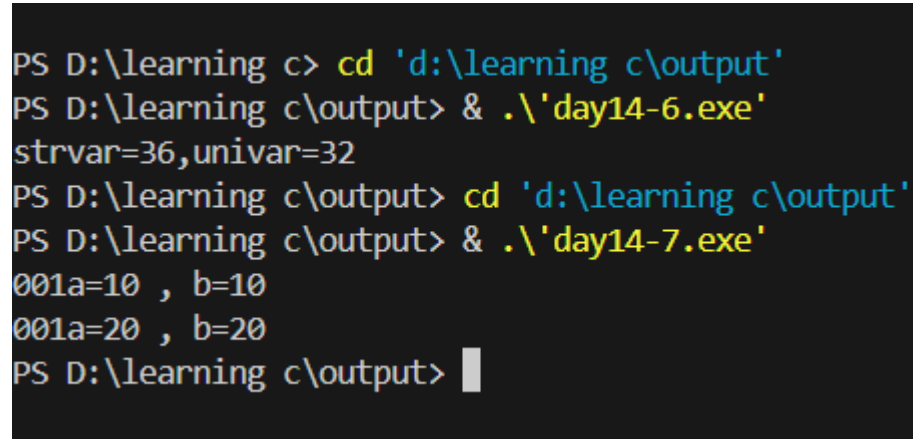
```
    printf("001a=%d , b=%d \n",u1.a,u1.b);
```

```
    u1.b=20;
```

```
    printf("001a=%d , b=%d \n",u1.a,u1.b);
```

```
    //since a , b share same memory location , at a time only one variable can be intialized
```

```
}
```



```
PS D:\learning c> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-6.exe'
strvar=36,univar=32
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-7.exe'
001a=10 , b=10
001a=20 , b=20
PS D:\learning c\output> █
```

8. //using pointers in union

```
//union
```

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

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

```
union course1
```

```
{
```

```
    int a;
```

```
    int b;
```

```
}u1;
```

```

int main()
{ union course1 *ptr;

  ptr=&u1;

  ptr->a=10;

  printf("001a=%d , b=%d \n",ptr->a,ptr->b);

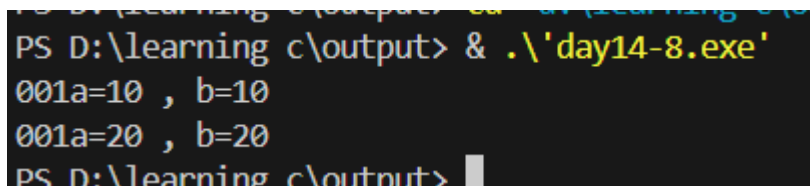
  ptr->b=20;

  printf("001a=%d , b=%d \n",ptr->a,ptr->b);

  //since a , b share same memory location , at a time only one variable can be intialized

}

```



```

PS D:\learning c\output> & .\'day14-8.exe\'
001a=10 , b=10
001a=20 , b=20
PS D:\learning c\output>

```

9. //accessing elements of a union

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

```

union Data {

  int a;

  int b;

}var;

```

```
void add(union Data);
```

```

int main() {

  union Data var;

  var.a = 10;

  printf("001 a: %d, b: %d\n", var.a, var.b);
}

```

```
add(var);
```

```
var.b = 20;
```

```
printf("002 a: %d, b: %d\n", var.a, var.b);
```

```
add(var);
```

```
return 0;
```

```
}
```

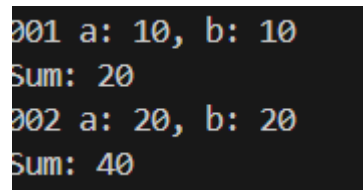
```
void add(union Data d) {
```

```
    int sum = 0;
```

```
    sum = d.a + d.b;
```

```
    printf("Sum: %d\n", sum);
```

```
}
```



```
001 a: 10, b: 10
Sum: 20
002 a: 20, b: 20
Sum: 40
```

10. //accessing elements of a union

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

```
union Data {
```

```
    int a;
```

```
    int b;
```

```
}var;
```

```
void add(union Data *);
```

```
int main() {
```

```

union Data *ptr = &var;

var.a = 10;

printf("001 a: %d, b: %d\n", var.a, var.b);

add(ptr);

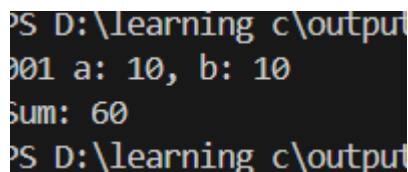
return 0;
}

```

```

void add(union Data *ptr1) {
    ptr1->a = 30;
    int sum = 0;
    sum = ptr1->a + ptr1->b;
    printf("Sum: %d\n", sum);
}

```



```

PS D:\learning c\output
001 a: 10, b: 10
Sum: 60
PS D:\learning c\output

```

### 11.Problem Statement: Vehicle Registration System

Write a C program to simulate a vehicle registration system using **unions** to handle different types of vehicles. The program should:

1. Define a union named Vehicle with the following members:
  - car\_model (character array of size 50): To store the model name of a car.
  - bike\_cc (integer): To store the engine capacity (in CC) of a bike.
  - bus\_seats (integer): To store the number of seats in a bus.
2. Create a structure VehicleInfo that contains:
  - type (character): To indicate the type of vehicle (C for car, B for bike, S for bus).
  - Vehicle (the union defined above): To store the specific details of the vehicle based on its type.
3. Implement the following features:
  - **Input Details:** Prompt the user to input the type of vehicle and its corresponding details:

- For a car: Input the model name.
  - For a bike: Input the engine capacity.
  - For a bus: Input the number of seats.
  - **Display Details:** Display the details of the vehicle based on its type.
4. Use the union effectively to save memory and ensure only relevant information is stored.

### Constraints

- The type of vehicle should be one of C, B, or S.
- For invalid input, prompt the user again.

```
#include<stdio.h>

union Vechicle
{
    char car_model[50];
    int bike_cc;
    int bus_seats;
};

struct Vechicleinfo
{
    char type;
    union Vechicle v;
};

int main()
{
    struct Vechicleinfo vi;

    printf("enter the type (C for Car , B for Bike , S for Bus) \n");
    scanf("%c",&vi.type);

    if(vi.type=='C')
    {
        printf("enter car model \n");
```

```
scanf("%s",vi.v.car_model);

}

else if(vi.type=='B')
{
    printf("enter bike engine capacity \n");
    scanf("%d",&vi.v.bike_cc);
}

else if(vi.type=='S')
{
    printf("enter no.of seats \n");
    scanf("%d",&vi.v.bus_seats);
}

else
{
    printf("invalid input \n");
}

if(vi.type=='C')
{
    printf("car model %s\n",vi.v.car_model);

}

else if(vi.type=='B')
{
    printf("bike engine capacity %d\n",vi.v.bike_cc);

}

else if(vi.type=='S')
{
    printf(" no.of seats %d \n",vi.v.bus_seats);
```

```
}  
return 0;  
  
}
```

```
enter the type (C for Car , B for Bike , S for Bus)  
C  
enter car model  
swift  
car model swift  
PS D:\learning c\output> cd 'd:\learning c\output'  
PS D:\learning c\output> & .\'day14-11.exe'  
enter the type (C for Car , B for Bike , S for Bus)  
B  
enter bike engine capacity  
150  
bike engine capacity 150  
PS D:\learning c\output> cd 'd:\learning c\output'  
PS D:\learning c\output> & .\'day14-11.exe'  
enter the type (C for Car , B for Bike , S for Bus)  
S  
enter no.of seats  
50  
no.of seats 50  
PS D:\learning c\output> █
```

12. //enum

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

```
enum math
```

```
{  
    add=1,  
    sub,  
    div
```

```
};
```

```
int main()
```

```
{
```



```

enum math var1=add;

printf("%d",var1);

enum math var2=sub;

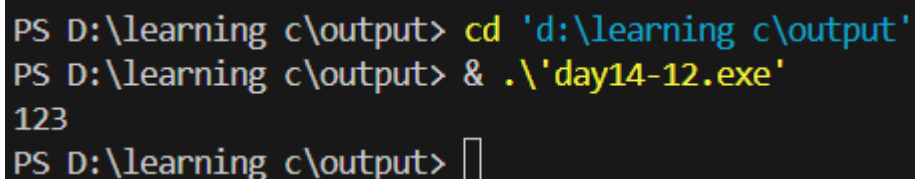
printf("%d",var2);


enum math var3=div;


printf("%d",var3);


return 0;
}

```



```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-12.exe'
123
PS D:\learning c\output> 

```

13. //enum

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

```
enum math
```

```
{
```

```
    add=10,
```

```
    sub=20,
```

```
    div=60
```

```
};
```

```
int main()
```

```
{
```

```
    enum math var1=add;
```

```
    printf("%d \n",var1);
```

```
    enum math var2=sub;
```

```
    printf("%d \n",var2);
```

```

    enum math var3=div;

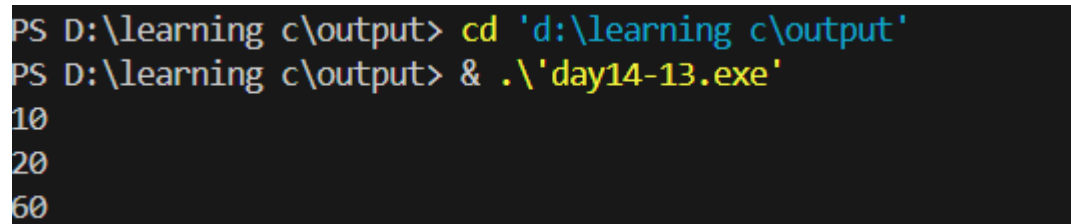
```

```

printf("%d \n",var3);

return 0;
}

```



```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-13.exe'
10
20
60

```

14. //enum-used for integer constants only

//if float given erroe

//if char , it's ascii is taken

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

```
enum math
```

```
{
```

```
    add=1,
```

```
    sub,
```

```
    div
```

```
};
```

```
int main()
```

```
{
```

```
    enum math var1=div;
```

```
    printf("size of var1 is %d \n",sizeof(var1));//size of enum is 4 byte
```

```
    switch(var1)
```

```
    {
```

```
        case 1:
```

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

```
        break;
```

```
        case 2:
```

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

```

        break;

    case 3:

        printf("division \n");

        break;

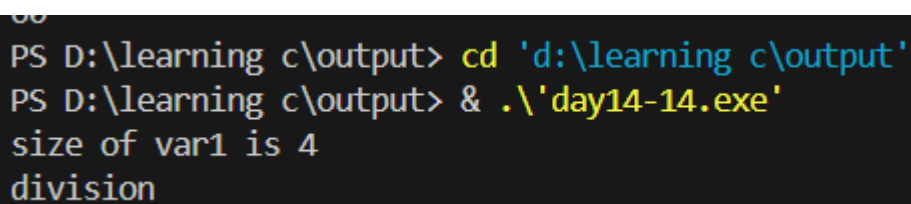
    default:

        break;

}

}

```



```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-14.exe'
size of var1 is 4
division

```

15.

### Problem 1: Traffic Light System

#### Problem Statement:

Write a C program to simulate a traffic light system using enum. The program should:

1. Define an enum named TrafficLight with the values RED, YELLOW, and GREEN.
2. Accept the current light color as input from the user (as an integer: 0 for RED, 1 for YELLOW, 2 for GREEN).
3. Display an appropriate message based on the current light:
  - RED: "Stop"
  - YELLOW: "Ready to move"
  - GREEN: "Go"

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

```
enum trafficLight
```

```
{
```

```
    red,
```

```
    yellow,
```

```
    green
```

```
};  
  
int main()  
{  
    int choice;  
    printf("enter : 0 for red light , 1 for yellow , 2 for green\n");  
    scanf("%d",&choice);  
    enum trafficLight tl=choice;  
    switch (tl)  
    {  
        case red:  
            printf("stop \n");  
            break;  
        case yellow:  
            printf("ready to move \n");  
            break;  
        case green:  
            printf("go \n");  
            break;  
  
        default:  
            break;  
    }  
  
}
```

```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-15.exe'
enter : 0 for red light , 1 for yellow , 2 for green
0
stop
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-15.exe'
enter : 0 for red light , 1 for yellow , 2 for green
1
ready to move
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-15.exe'
enter : 0 for red light , 1 for yellow , 2 for green
2
go
PS D:\learning c\output> █

```

16,

## Problem 2: Days of the Week

### Problem Statement:

Write a C program that uses an enum to represent the days of the week. The program should:

1. Define an enum named Weekday with values MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, and SUNDAY.
2. Accept a number (1 to 7) from the user representing the day of the week.
3. Print the name of the day and whether it is a weekday or a weekend.
  - Weekends: SATURDAY and SUNDAY
  - Weekdays: The rest

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

```
enum Days
```

```
{
```

```
    monday=1,
```

```
    tuesday,
```

```
    wednesday,
```

```
    thursday,
```

```
    friday,  
    saturday,  
    sunday  
};  
int main()  
{  
    int day;  
    printf("enter a day (1 to 7 )\n");  
    scanf("%d",&day);  
    enum Days d=day;  
    switch(d)  
    {  
        case monday:  
            printf(" monday :%d \n",d);  
            printf("it is a weekday \n");  
            break;  
        case tuesday:  
            printf(" tuesday :%d \n",d);  
            printf("it is weekday \n");  
            break;  
        case wednesday:  
            printf(" wednesday:%d \n",d);  
            printf("it is a weekday \n");  
            break;  
        case thursday:  
            printf("thursday:%d \n",d);  
            printf("it is weekday \n");  
            break;  
        case friday:  
            printf("friday:%d \n",d);  
            printf("it is a weekday \n");
```

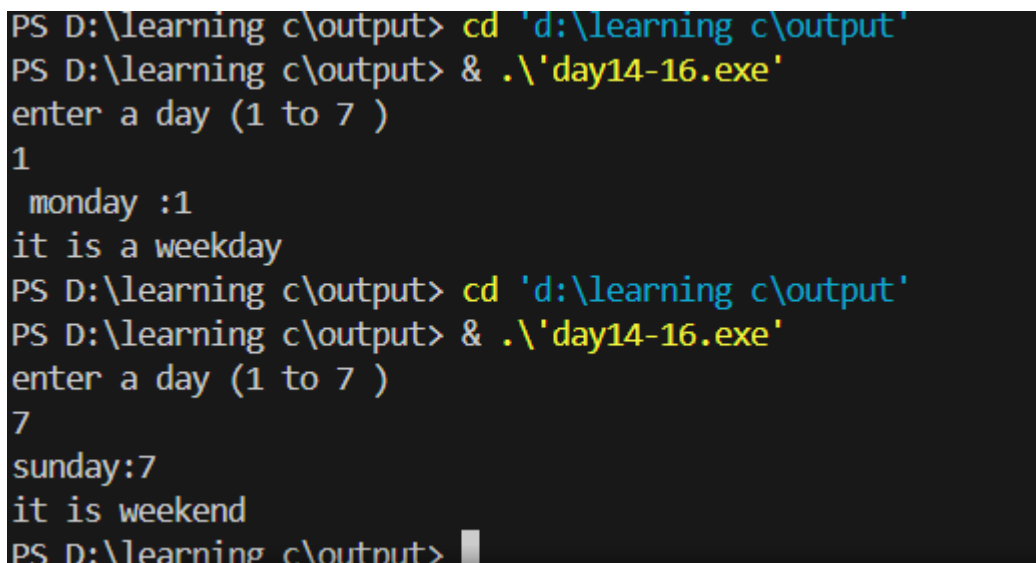
```

        break;
case saturday:
    printf(" sat:%d \n",d);
    printf("it is weekend \n");
    break;
case sunday:
    printf("sunday:%d \n",d);
    printf("it is weekend \n");
    break;

}

return 0;
}

```



```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-16.exe'
enter a day (1 to 7 )
1
monday :1
it is a weekday
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-16.exe'
enter a day (1 to 7 )
7
sunday:7
it is weekend
PS D:\learning c\output>

```

### Problem 3: Shapes and Their Areas

#### Problem Statement:

Write a C program to calculate the area of a shape based on user input using enum. The program should:

1. Define an enum named Shape with values CIRCLE, RECTANGLE, and TRIANGLE.
2. Prompt the user to select a shape (0 for CIRCLE, 1 for RECTANGLE, 2 for TRIANGLE).
3. Based on the selection, input the required dimensions:
  - For CIRCLE: Radius

- For RECTANGLE: Length and breadth
  - For TRIANGLE: Base and height
4. Calculate and display the area of the selected shape.

```
#include<stdio.h>

enum Shape

{
    circle,
    rectangle,
    triangle
};

int main()
{
    int value;

    printf("enter a number 0 for circle , 1 for rectangle , 2 for traingle \n");
    scanf("%d",&value);
    enum Shape sh=value;

    int r,l,br,bs,h;

    int area;

    switch (sh)
    {
        case circle:
            printf("circle \n");
            printf("enter radius \n");
            scanf("%d",&r);
            area=3.14*r*r;
            printf("area is %d \n",area);

            break;
        case rectangle:
            printf("rectangle \n");
            printf("enter length\n");
```



```
scanf("%d",&l);  
printf("enter breadth\n");  
scanf("%d",&br);  
  
area=l*br;  
printf("area is %d \n",area);  
  
break;  
case triangle:  
    printf("triangle \n");  
    printf("enter base \n");  
    scanf("%d",&bs);  
    printf("enter height\n");  
    scanf("%d",&h);  
    area=0.5*bs*h;  
    printf("area is %d \n",area);  
  
break;  
  
default:  
    break;  
}  
return 0;  
}
```

```

enter a number 0 for circle , 1 for rectangle , 2 for traingle
0
circle
enter radius
2
area is 12
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-17.exe'
enter a number 0 for circle , 1 for rectangle , 2 for traingle
1
rectangle
enter length
2
enter breadth
3
area is 6
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-17.exe'
enter a number 0 for circle , 1 for rectangle , 2 for traingle
2
triangle
enter base
2
enter height
3

```

#### Problem 4: Error Codes in a Program

##### Problem Statement:

Write a C program to simulate error handling using enum. The program should:

1. Define an enum named ErrorCode with values:
  - SUCCESS (0)
  - FILE\_NOT\_FOUND (1)
  - ACCESS\_DENIED (2)
  - OUT\_OF\_MEMORY (3)
  - UNKNOWN\_ERROR (4)
2. Simulate a function that returns an error code based on a scenario.
3. Based on the returned error code, print an appropriate message to the user.

```
#include<stdio.h>

enum ErrorCode
{
    sucess,
    file_not_found,
    acess_denied,
    out_of_memory,
    unknown_error
};

int main()
{
    int v;
    printf("enter a value 0to 4 \n");
    scanf("%d",&v);
    enum ErrorCode ec=v;
    switch (ec)
    {
    case 0:
        printf("sucess(%d)\n",ec);
        break;
    case 1:
        printf("file_not_found(%d)\n",ec);
        break;
    case 2:
        printf("acess denied(%d)\n",ec);
        break;
    case 3:
        printf("out of memory(%d)\n",ec);
        break;
    case 4:
        printf("unknown error(%d)\n",ec);
```

```

        break;

default:
    break;
}

return 0;
}

```

```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-18.exe'
enter a value 0to 4
0
sucess(0)
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-18.exe'
enter a value 0to 4
1
file_not_found(1)
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-18.exe'
enter a value 0to 4
2
acess denied(2)
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-18.exe'
enter a value 0to 4
3
out of memory(3)
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-18.exe'
enter a value 0to 4
4
unknown error(4)
PS D:\learning c\output>

```

## Problem 5: User Roles in a System

### Problem Statement:

Write a C program to define user roles in a system using enum. The program should:

1. Define an enum named UserRole with values ADMIN, EDITOR, VIEWER, and GUEST.

2. Accept the user role as input (0 for ADMIN, 1 for EDITOR, etc.).
3. Display the permissions associated with each role:
  - ADMIN: "Full access to the system."
  - EDITOR: "Can edit content but not manage users."
  - VIEWER: "Can view content only."
  - GUEST: "Limited access, view public content only."
  -

```
#include<stdio.h>

enum userRole
{
    admin,
    editor,
    viewer,
    guest
};

int main()
{
    int v;
    printf("enter a value 0to 3 \n");
    scanf("%d",&v);
    enum userRole ur=v;
    switch (ur)
    {
    case 0:
        printf("admin:%d\n",ur);
        printf("full access to the system \n");
        break;
    case 1:
        printf("editor:%d\n",ur);
```

```
    printf("can edit content but not manage users \n");
    break;
case 2:
    printf("viewer:%d\n",ur);
    printf("can view content only \n");
    break;
case 3:
    printf("guest:%d\n",ur);
    printf("limited access , view public content only \n");
    break;
case 4:
    printf("enter valid input \n");
    break;

default:
    break;
}
return 0;
}
```

```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-19.exe'
enter a value 0to 3
0
admin:0
full access to the system
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-19.exe'
enter a value 0to 3
1
editor:1
can edit content but not manage users
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-19.exe'
enter a value 0to 3
2
viewer:2
can view content only
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-19.exe'
enter a value 0to 3
3
guest:3
limited access , view public content only
PS D:\learning c\output>

```

20. //bit fields

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

```
struct date
```

```
{
```

```
    int day:5;
```

```
    int month:4;
```

```
    int year;
```

```
};
```

```
int main()
```

```
{
```

```
    printf("size of date is%d \n",sizeof(struct date));
```

```
    struct date d1={25,11,2024};
```

```
    printf("date is %d-%d-%d \n",d1.day,d1.month,d1.year);  
    return 0;  
}
```

21. //bit fields

```
#include<stdio.h>  
  
struct date  
{  
    unsigned int day:5;  
    unsigned int month:4;  
    unsigned int year;  
};  
  
int main()  
{  
    printf("size of date is%d \n",sizeof(struct date));  
    struct date d1={25,11,2024};  
    printf("date is %d-%d-%d \n",d1.day,d1.month,d1.year);  
    return 0;  
}
```

22. //forced alginment

```
#include<stdio.h>  
  
struct date  
{  
    int d:5;  
    int :0;  
    int y:8;  
};  
  
int main()  
{  
    printf("size of date is%d \n",sizeof(struct date));  
  
}
```



```

23. #include<stdio.h>

struct t1
{
    unsigned int d:5;//cannot access adress of bitfileds

    unsigned int m:4;//there pointers can't be used
    int year;
    int a[10]:5
    //array of bitfileds is not allowed
};

int main()
{
    struct t1 test;
    printf("adress of d=%p , address of m=%p",test.d,test.m);

}

```

### Problem 1: Compact Date Storage

#### Problem Statement:

Write a C program to store and display dates using bit-fields. The program should:

Define a structure named Date with bit-fields:

day (5 bits): Stores the day of the month (1-31).

month (4 bits): Stores the month (1-12).

year (12 bits): Stores the year (e.g., 2024).

Create an array of dates to store 5 different dates.

Allow the user to input 5 dates in the format DD MM YYYY and store them in the array.

Display the stored dates in the format DD-MM-YYYY.

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

```
struct Date {
```

```
    unsigned int day: 5;
```

```
    unsigned int month: 4;
```

```
    unsigned int year: 12;
```

```
};
```

```
int main() {
```

```
    struct Date arr[5];
```

```
    printf("Enter the dates:\n");
```

```
    for (int i = 0; i < 5; i++) {
```

```
        unsigned int temp;
```

```
        printf("Enter day: ");
```

```
        scanf("%d",&temp);
```

```
        arr[i].day=temp;
```

```
        printf("Enter month: ");
```

```
        scanf("%d",&temp);
```

```
        arr[i].month=temp;
```

```
        printf("Enter year: ");
```

```
        scanf("%d",&temp);
```

```
        arr[i].year=temp;
```

```
    }
```

```
// Optional: Print the entered dates for verification
```

```
printf("\nEntered dates are:\n");
```

```
for (int i = 0; i < 5; i++) {
```

```
    printf("Date %d: %d/%d/%d\n", i + 1, arr[i].day, arr[i].month, arr[i].year);
```

```
}
```

```
    return 0;  
}
```

```
PS D:\learning c\output> & .\'day14-24.exe'  
Enter the dates:  
Enter day: 2  
Enter month: 11  
Enter year: 24  
Enter day: 3  
Enter month: 11  
Enter year: 24  
Enter day: 4  
Enter month: 11  
Enter year: 24  
Enter day: 5  
Enter month: 11  
Enter year: 24  
Enter day: 6  
Enter month: 11  
Enter year: 24  
  
Entered dates are:  
Date 1: 2/11/24  
Date 2: 3/11/24  
Date 3: 4/11/24  
Date 4: 5/11/24  
Date 5: 6/11/24
```

## Problem 2: Status Flags for a Device

### Problem Statement:

Write a C program to manage the status of a device using bit-fields. The program should:

Define a structure named `DeviceStatus` with the following bit-fields:

`power` (1 bit): 1 if the device is ON, 0 if OFF.

`connection` (1 bit): 1 if the device is connected, 0 if disconnected.

`error` (1 bit): 1 if there's an error, 0 otherwise.

Simulate the device status by updating the bit-fields based on user input:

Allow the user to set or reset each status.

Display the current status of the device in a readable format (e.g., Power: ON, Connection: DISCONNECTED, Error: NO).

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

```
struct DeviceStatus
```

```
{
```

```
    unsigned int power:1;
```

```
    unsigned int connection:1;
```

```
    unsigned int error:1;
```

```
};
```

```
int main()
```

```
{
```

```
    struct DeviceStatus ds;
```

```
    unsigned int choice;
```

```
    printf("enter a power value 0 Or 1");
```

```
    scanf("%d",&choice);
```

```
    ds.power=choice;
```

```
    if(ds.power==1)
```

```
    {
```

```
        printf("power: on \n");
```

```
    }
```

```
    else if(ds.power==0)
```

```
    {
```

```
        printf("power:off \n");
```

```
    }
```

```
    else
```

```
    {
```

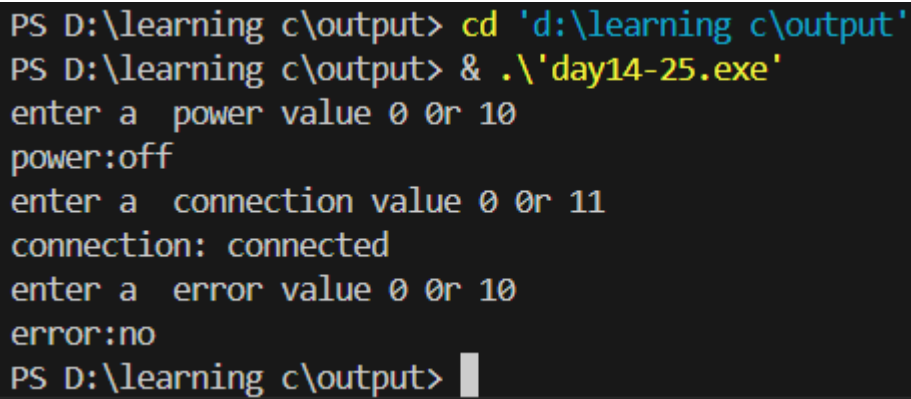
```
        printf("enter a valid input\n");
```

```
    }
```

```
    printf("enter a connection value 0 Or 1");
```

```
scanf("%d",&choice);
ds.connection=choice;
if(ds.connection==1)
{
    printf("connection: connected\n");
}
else if(ds.connection==0)
{
    printf("connection:disconnected \n");
}
else
{
    printf("enter a valid input\n");
}
printf("enter a error value 0 Or 1");
scanf("%d",&choice);
ds.error=choice;
if(ds.error==1)
{
    printf("error: yes \n");
}
else if(ds.error==0)
{
    printf("error:no \n");
}
else
{
    printf("enter a valid input\n");
}
return 0;
```

```
}
```

A screenshot of a Windows command prompt window. The background is black, and the text is in a monospaced font with some color coding. The prompt shows the user navigating to 'D:\learning c\output' and running a program named 'day14-25.exe'. The program prompts the user for a power value (0 or 10), a connection value (0 or 11), and an error value (0 or 10). The user has entered 'power:off', 'connection: connected', and 'error:no'. The prompt is currently at 'PS D:\learning c\output>' with a cursor.

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-25.exe'
enter a power value 0 or 10
power:off
enter a connection value 0 or 11
connection: connected
enter a error value 0 or 10
error:no
PS D:\learning c\output> █
```

### Problem 3: Storage Permissions

#### Problem Statement:

Write a C program to represent file permissions using bit-fields. The program should:

Define a structure named FilePermissions with the following bit-fields:

read (1 bit): Permission to read the file.

write (1 bit): Permission to write to the file.

execute (1 bit): Permission to execute the file.

Simulate managing file permissions:

Allow the user to set or clear each permission for a file.

Display the current permissions in the format R:1 W:0 X:1 (1 for permission granted, 0 for denied).

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

```
struct FilePermissions
```

```
{
```

```
    unsigned int read:1;
```

```
    unsigned int write:1;
```

```
    unsigned int execute:1;
```

```
};
```

```
int main()
```

```
{
```

```
    struct FilePermissions file;
```

```

unsigned int p;

printf("enter read :\n");

scanf("%d",&p);

file.read=p;

printf("enter write :\n");

scanf("%d",&p);

file.write=p;

printf("enter execute :\n");

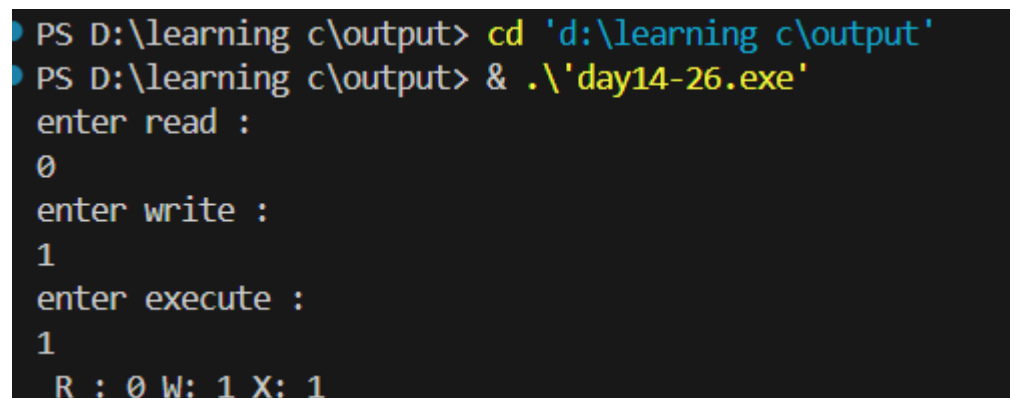
scanf("%d",&p);

file.execute=p;

printf(" R : %d W: %d X: %d",file.read,file.write,file.execute);


return 0;
}

```



```

PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day14-26.exe'
enter read :
0
enter write :
1
enter execute :
1
R : 0 W: 1 X: 1

```

#### Problem 4: Network Packet Header

##### Problem Statement:

Write a C program to represent a network packet header using bit-fields. The program should:

Define a structure named PacketHeader with the following bit-fields:

version (4 bits): Protocol version (0-15).

IHL (4 bits): Internet Header Length (0-15).

type\_of\_service (8 bits): Type of service.

total\_length (16 bits): Total packet length.

Allow the user to input values for each field and store them in the structure.

```

#include<stdio.h>

struct PacketHeader
{
    unsigned int version:4;
    unsigned int IHL:4;
    unsigned int type_of_service:8;
    unsigned int total_length:16;
};

int main()
{
    struct PacketHeader ph;
    unsigned int p;
    printf("enter version :\n");
    scanf("%d",&p);
    ph.version=p;
    printf("enter IHL :\n");
    scanf("%d",&p);
    ph.IHL=p;
    printf("enter type of service :\n");
    scanf("%d",&p);
    ph.type_of_service=p;
    printf("enter total length \n");
    scanf("%d",&p);
    ph.total_length=p;
    printf("packet header details \n");
    printf("version %d \n",ph.version);
    printf("IHL %d \n",ph.IHL);
    printf("type of service :%d \n",ph.type_of_service);
    printf("total length %d \n",ph.total_length);
    return 0;
}

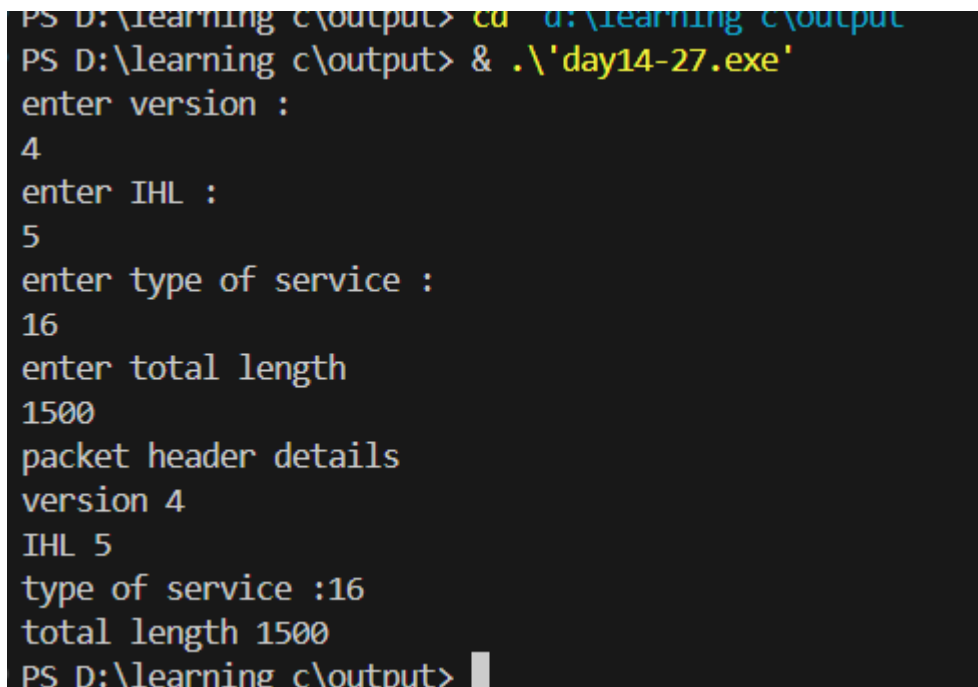
```



```

}
/*Enter version (0-15):
4
Enter IHL (0-15):
5
Enter type of service (0-255):
16
Enter total length (0-65535):
1500
*/

```



```

PS D:\learning c\output> cd d:\learning c\output
PS D:\learning c\output> & .\'day14-27.exe'
enter version :
4
enter IHL :
5
enter type of service :
16
enter total length
1500
packet header details
version 4
IHL 5
type of service :16
total length 1500
PS D:\learning c\output>

```

Display the packet header details in a structured format.

## Problem 5: Employee Work Hours Tracking

Problem Statement:

Write a C program to track employee work hours using bit-fields. The program should:

Define a structure named WorkHours with bit-fields:

days\_worked (7 bits): Number of days worked in a week (0-7).

hours\_per\_day (4 bits): Average number of hours worked per day (0-15).

Allow the user to input the number of days worked and the average hours per day for an employee.

Calculate and display the total hours worked in the week.

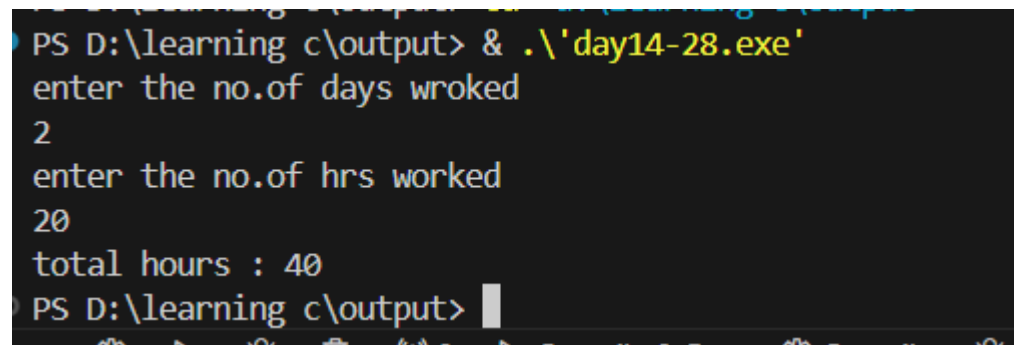
```
#include<stdio.h>

struct WorkHours
{
    unsigned int days_worked;
    unsigned int hours_per_day;

};

int main()
{
    struct WorkHours wh;
    unsigned int temp;
    unsigned int total_hrs;
    printf("enter the no.of days wroked \n");
    scanf("%d",&temp);
    wh.days_worked=temp;
    printf("enter the no.of hrs worked \n");
    scanf("%d",&temp);
    wh.hours_per_day=temp;
    total_hrs=wh.days_worked*wh.hours_per_day;
    printf("total hours : %d \n",total_hrs);

    return 0;
}
```



```
PS D:\learning c\output> & .\'day14-28.exe'
enter the no.of days wroked
2
enter the no.of hrs worked
20
total hours : 40
PS D:\learning c\output>
```