PHASE 1 DAY 15

```
/*
typedef is a keyword: that is used to provide an alias

or a new name to an already existing data type


C syntax for typedef


typedef existing_name_of_the_data_type alias_name;


example

typedef double dbl;
*/

#include <stdio.h>
typedef int my_int;
int main()
{
	//alias name my_int has been used for declaring the variable
	my_int a = 28;

	printf("a = %d\n", a);

	return 0;
}
```

TYPEDEF WITH STRUCTURES

```
#include <stdio.h>
typedef struct
{
    int date;
    int month;
    int year;
}dt;
int main()
{
    dt var={26,10,2024};

    printf("%d/%d/%d \n",var.date,var.month,var.year);
    printf("size=%d",sizeof(var));

    return 0;
}
```

TYPEDEF WITH POINTERS

```
#include <stdio.h>
typedef int * intPtr;
int main()
{
int a=30;
    intPtr ptr1=&a;
    printf("001a =%d\n",*ptr1);
    *ptr1=30;
    printf("002a =%d\n",*ptr1);
    return 0;
}
```

TYPEDEF WITH ARRAYS

```
#include <stdio.h>
typedef int arr[4];
int main()
{
arr t={1,2,3,4};
   for(int i=0;i<4;i++)
   {
   printf("%d\n",t[i]);
   }
   return 0;
}
```

Problem Statement:
Write a program that defines a custom data type Complex using typedef to represent a complex number with real and imaginary parts. Implement functions to:

- Add two complex numbers.
- Multiply two complex numbers.
- Display a complex number in the format "a + bi".

Input Example

Enter first complex number (real and imaginary): 3 4

Enter second complex number (real and imaginary): 1 2

Output Example

Sum: 4 + 6i

Product: -5 + 10i

Answer:

```
#include <stdio.h>
typedef int complex[2];

void add(complex,complex);
void product(complex,complex);

int main()
{
   complex num1,num2;
```

```
  printf("enter num1:");
  scanf("%d %d",&num1[0],&num1[1]);
  printf("enter num2:");
  scanf("%d %d",&num2[0],&num2[1]);

  add(num1,num2);
  product(num1,num2);

    return 0;
}
void add(complex n1,complex n2)
 {
    int real=n1[0]+n2[0];
    int imag=n1[1]+n2[1];
    printf("Sum:%d+%di",real,imag);
 }
void product(complex n1, complex n2)
{
    int real=((n1[0]*n2[0])-(n1[1]*n2[1]));
    int imag=((n1[0]*n2[1])+(n1[1]*n2[0]));
    printf("\nProduct:%d+%di",real,imag);

}
```

Typedef for Structures

Problem Statement:
Define a custom data type Rectangle using typedef to represent a rectangle with width and height as float values. Write functions to:

- Compute the area of a rectangle.
- Compute the perimeter of a rectangle.

Input Example:

Enter width and height of the rectangle: 5 10

Output Example:

Area: 50.00

Perimeter: 30.00

Answer:

```
#include <stdio.h>
typedef float rectangle[2];

void area(rectangle);
void perimeter(rectangle);

int main()
{
    rectangle rect;

  printf("enter length and bredth:");
  scanf("%f %f",&rect[0],&rect[1]);

  area(rect);
  perimeter(rect);

    return 0;
}
void area(rectangle r)
 {
    printf("\nArea:%.2f",r[0]*r[1]);
 }
void perimeter(rectangle r)
{
    printf("\nPerimeter:%.2f",(2*(r[0]+r[1])));

}
```

*Funtion Pointers */

```
#include <stdio.h>

 void display(int);
```

```
 int main()

{
        //Declaration a pointer to the function display()
        void (*func_ptr)(int);
        //Initializing the pointer with the address of function display()
        func_ptr = &display;
        //Calling the function as well passing the parameter using function pointers
        (*func_ptr)(20);
        return 0;
 }

 void display(int a)

{
        printf("a = %d",a);
 }


//array of function pointers

#include <stdio.h>

void add(int, int);
void sub(int, int);
void mul(int, int);

int main()
 {
   void (*fun_ptr_arr[])(int, int) = {add, sub, mul};

   int a = 10, b = 20;

   (*fun_ptr_arr[0])(a, b);
   (*fun_ptr_arr[1])(a, b);
   (*fun_ptr_arr[2])(a, b);

   return 0;
}

void add(int x, int y)
```

```
{
   printf("%d + %d = %d\n", x, y, x + y);
}

void sub(int x, int y)
{
   printf("%d - %d = %d\n", x, y, x - y);
}

void mul(int x, int y)
{
   printf("%d * %d = %d\n", x, y, x * y);
}
```

1.Simple Calculator Using Function Pointers
Problem Statement:
Write a C program to implement a simple calculator. Use function pointers to dynamically call functions for addition, subtraction, multiplication, and division based on user input.
Input Example:
Enter two numbers: 10 5
Choose operation (+, -, *, /): *
Output Example:
Result: 50

Answer:

```
#include <stdio.h>

void add(int, int);
void sub(int, int);
void mul(int, int);
void divi(int,int);

int main()
{
    int n1,n2;
    char ch;
```

```
    void (*fun_ptr_arr[])(int, int) = {add, sub, mul, divi };


    printf("\nEnter 2 numbers :");
    scanf("%d %d",&n1,&n2);

    printf("\nEnter the operator (+,-,*,/):");
    scanf(" %c",&ch);

    switch(ch)
    {
        case '+':
        {
           (*fun_ptr_arr[0])(n1, n2);
           break;
        }
         case '-':
        {
            (*fun_ptr_arr[1])(n1, n2);
            break;
        }
         case '*':
        {
            (*fun_ptr_arr[2])(n1, n2);
           break;
        }
         case '/':
        {
            (*fun_ptr_arr[3])(n1, n2);
             break;
        }
       default:
       printf("Invalid");
    }

    return 0;
}

void add(int x, int y)
 {
```

```
    printf("%d + %d = %d\n", x, y, x + y);
}

void sub(int x, int y)
 {
    printf("%d - %d = %d\n", x, y, x - y);
}

void mul(int x, int y)
 {
    printf("%d * %d = %d\n", x, y, x * y);
}
void divi(int x, int y)
 {
    printf("%d / %d = %d\n", x, y, x / y);
}
```

2.Array Operations Using Function Pointers
Problem Statement:
Write a C program that applies different operations to an array of integers using function pointers. Implement operations like finding the maximum, minimum, and sum of elements.
Input Example:
Enter size of array: 4
Enter elements: 10 20 30 40
Choose operation (1 for Max, 2 for Min, 3 for Sum): 3
Output Example:
Result: 100

Answer:

```
#include <stdio.h>

void Max(int*,int);
void Min(int *,int);
void Sum(int *,int);

int main()
```

```
{
    int n;
    int ch;

   void (*fun_ptr_arr[])(int *,int) = {Max,Min,Sum};


   printf("\nEnter the size :");
   scanf("%d",&n);
   int arr[n];
    printf("\nEnter elements into the array:");
   for(int j=0;j<n;j++)
   {
       scanf("%d",&arr[j]);
   }

   printf("\nChoose operation (1-Max 2-Min 3-Sum):");
   scanf("%d",&ch);


   switch(ch)
   {
       case 1:
       {
          (*fun_ptr_arr[0])(arr,n);
          break;
       }
        case 2:
       {
           (*fun_ptr_arr[1])(arr,n);
           break;
       }
        case 3:
       {
           (*fun_ptr_arr[2])(arr,n);
          break;
       }
      default:
      printf("Invalid");
   }
```

```
    return 0;
}

void Max(int*arr,int n)
 {
    int max=arr[0];
    for(int i=0;i<n;i++)
    {
        if(arr[i]>max)
        {
            max=arr[i];
        }
    }
    printf("Maximum:%d",max);
 }

void Min(int*arr,int n)
 {
     int min=arr[0];
    for(int i=0;i<n;i++)
    {
        if(arr[i]<min)
        {
            min=arr[i];
        }
    }
    printf("Minumum:%d",min);
 }

void Sum(int*arr,int n)
 {
     int sum=0;
    for(int i=0;i<n;i++)
    {
      sum += arr[i];
    }
    printf("Sum:%d",sum);
}
```

3.Event System Using Function Pointers
Problem Statement:
Write a C program to simulate a simple event system. Define three events: onStart, onProcess, and onEnd. Use function pointers to call appropriate event handlers dynamically based on user selection.
Input Example:
Choose event (1 for onStart, 2 for onProcess, 3 for onEnd): 1
Output Example:
Event: onStart
Starting the process...

Answer:

```
#include <stdio.h>

void onStart();
void onProcess();
void onEnd();

int main()
 {

    int ch;


   printf("\nChoose operation (1-onStart 2-onProcess 3-onEnd):");
   scanf("%d",&ch);
      void (*fun_ptr_arr[])() = {onStart,onProcess,onEnd};

   switch(ch)
   {
      case 1:
      {
        (*fun_ptr_arr[0])();
        break;
      }
       case 2:
      {
```

```
            (*fun_ptr_arr[1])();
            break;
        }
         case 3:
        {
            (*fun_ptr_arr[2])();
           break;
        }
       default:
       printf("Invalid");
    }

    return 0;
}

void onStart()
 {
    printf("Starting the process..");
 }

void onProcess()
 {
    printf("process running..");
 }

void onEnd()
 {
    printf("ending process...");
 }
```

4.Matrix Operations with Function Pointers
Problem Statement:
Write a C program to perform matrix operations using function pointers. Implement functions to add, subtract, and multiply matrices. Pass the function pointer to a wrapper function to perform the desired operation.
Input Example:
Enter matrix size (rows and columns): 2 2
Enter first matrix:
1 2

3 4
Enter second matrix:
5 6
7 8
Choose operation (1 for Add, 2 for Subtract, 3 for Multiply): 1
Output Example:
Result:
6 8
10 12

```
#include <stdio.h>

void add(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int result[rows][cols]);
void subtract(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int
result[rows][cols]);
void multiply(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int
result[rows][cols]);
void wrapper(int rows, int cols,void (*operation)(int, int, int[][cols], int[][cols], int[][cols]),int
mat1[rows][cols], int mat2[rows][cols]);

int main()
{
   int rows, cols, choice;

   printf("Enter matrix size (rows and columns): ");
   scanf("%d %d", &rows, &cols);

   int mat1[rows][cols], mat2[rows][cols], result[rows][cols];

   printf("Enter first matrix:\n");
   for (int i = 0; i < rows; i++)
 {
       for (int j = 0; j < cols; j++)
 {
           scanf("%d", &mat1[i][j]);
       }
   }

   printf("Enter second matrix:\n");
   for (int i = 0; i < rows; i++)
{
       for (int j = 0; j < cols; j++)
```

```
{
            scanf("%d", &mat2[i][j]);
        }
    }

    printf("Choose operation (1 for Add, 2 for Subtract, 3 for Multiply): ");
    scanf("%d", &choice);

    void (*operations[])(int, int, int[][cols], int[][cols], int[][cols]) = {add, subtract, multiply};

    if (choice >= 1 && choice <= 3)
{
        wrapper(rows, cols, operations[choice - 1], mat1, mat2);
    }
else
{
        printf("Invalid choice!\n");
    }

    return 0;
}

void add(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int result[rows][cols])
{
    for (int i = 0; i < rows; i++)
{
        for (int j = 0; j < cols; j++)
{
            result[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
    printf("Result:\n");
    for (int i = 0; i < rows; i++)
{
        for (int j = 0; j < cols; j++)
{
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
}
```

```
void subtract(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int
result[rows][cols])
 {
   for (int i = 0; i < rows; i++)
{
       for (int j = 0; j < cols; j++)
{
           result[i][j] = mat1[i][j] - mat2[i][j];
       }
   }
   printf("Result:\n");
   for (int i = 0; i < rows; i++)
 {
       for (int j = 0; j < cols; j++)
 {
           printf("%d ", result[i][j]);
       }
       printf("\n");
   }
}

void multiply(int rows, int cols, int mat1[rows][cols], int mat2[rows][cols], int
result[rows][cols])
{
   if (rows != cols)
 {
       printf("Matrix multiplication is only supported for square matrices.\n");
       return;
   }
   for (int i = 0; i < rows; i++)
 {
       for (int j = 0; j < cols; j++)
 {
           result[i][j] = 0;
           for (int k = 0; k < cols; k++)
 {
               result[i][j] += mat1[i][k] * mat2[k][j];
           }
       }
   }
   printf("Result:\n");
   for (int i = 0; i < rows; i++)
```

```
{
      for (int j = 0; j < cols; j++)
{
          printf("%d ", result[i][j]);
      }
      printf("\n");
   }
}

void wrapper(int rows, int cols,  void (*operation)(int, int, int[][cols], int[][cols], int[][cols]),
          int mat1[rows][cols], int mat2[rows][cols])
{
   int result[rows][cols];
   operation(rows, cols, mat1, mat2, result);
}
```

Problem Statement: Vehicle Management System

Write a C program to manage information about various vehicles. The program should demonstrate the following:

1. **Structures**: Use structures to store common attributes of a vehicle, such as vehicle type, manufacturer name, and model year.
2. **Unions**: Use a union to represent type-specific attributes, such as:
    - Car: Number of doors and seating capacity.
    - Bike: Engine capacity and type (e.g., sports, cruiser).
    - Truck: Load capacity and number of axles.
3. **Typedefs**: Define meaningful aliases for complex data types using typedef (e.g., for the structure and union types).
4. **Bitfields**: Use bitfields to store flags for vehicle features like **airbags**, **ABS**, and **sunroof**.
5. **Function Pointers**: Use a function pointer to dynamically select a function to display specific information about a vehicle based on its type.

Requirements

1. Create a structure Vehicle that includes:
    - A char array for the manufacturer name.
    - An integer for the model year.
    - A union VehicleDetails for type-specific attributes.

    - A bitfield to store vehicle features (e.g., airbags, ABS, sunroof).
    - A function pointer to display type-specific details.
2. Write functions to:
    - Input vehicle data, including type-specific details and features.
    - Display all the details of a vehicle, including the type-specific attributes.
    - Set the function pointer based on the vehicle type.
3. Provide a menu-driven interface to:
    - Add a vehicle.
    - Display vehicle details.
    - Exit the program.

Example Input/Output

Input:

1. Add Vehicle
2. Display Vehicle Details
3. Exit
Enter your choice: 1

Enter vehicle type (1: Car, 2: Bike, 3: Truck): 1
Enter manufacturer name: Toyota
Enter model year: 2021
Enter number of doors: 4
Enter seating capacity: 5
Enter features (Airbags[1/0], ABS[1/0], Sunroof[1/0]): 1 1 0

1. Add Vehicle
2. Display Vehicle Details
3. Exit
Enter your choice: 2

Output:

Manufacturer: Toyota

Model Year: 2021

Type: Car

Number of Doors: 4

Seating Capacity: 5

Features: Airbags: Yes, ABS: Yes, Sunroof: No

Answer:

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

void add_vehicle();
void Display_vehicle();

void display_bike(struct VEHICLE *v);
void display_car(struct VEHICLE *v);
void display_truck(struct VEHICLE *v);

struct vehicle_features
{
   unsigned int airbags:1;
   unsigned int ABS:1;
   unsigned int sunroof:1;
};

union vehicle_details
{
   struct
   {
      int no_of_doors;
      int seats;
   }car;

   struct
   {
      int engine_capacity;
      char type[30];
   }bike;

   struct
   {
```

```
        int load_capacity;
        int no_of_axels;
    }truck;
};

struct VEHICLE
{
    char manufacturer_name[40];
    int model_year;
    union vehicle_details details;
    struct vehicle_features features;
    int vehicle_type;
    void(*fptr)(struct VEHICLE *v);
};
struct VEHICLE v;

int main()
{
    int choice;
    while(1)
    {
        printf("Choose option:\n1.Add Vehicle\n2.Display vehicle details.\n3.Exit\nEnter your
choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                add_vehicle();
                break;
            case 2:
                Display_vehicle();
                break;
            case 3:
                exit(0);
                break;
            default:
                printf("Invalid\n");
        }
    }

    return 0;
```

```
}

void add_vehicle()
{
   printf("\nEnter the type of vehicle (1: Car, 2: Bike, 3: Truck): ");
   scanf("%d", &v.vehicle_type);

   printf("Enter manufacturer name: ");
   scanf("%[^\n]",v.manufacturer_name);
      getchar();

   printf("Enter model year: ");
   scanf("%d", &v.model_year);

   switch(v.vehicle_type)
   {
      case 1:
         printf("Enter details of Car:\n");
         printf("Enter the number of doors: ");
         scanf("%d", &v.details.car.no_of_doors);

         printf("Enter the seating capacity: ");
         scanf("%d", &v.details.car.seats);

         printf("Enter features (Airbags[1/0], ABS[1/0], Sunroof[1/0]): ");
         scanf("%d %d %d", &v.features.airbags, &v.features.ABS, &v.features.sunroof);
         v.fptr = display_car;
         break;

      case 2:
         printf("Enter details of Bike:\n");
         printf("Enter the engine capacity: ");
         scanf("%d", &v.details.bike.engine_capacity);

         getchar();
         printf("Enter the type: ");
         fgets("%[^\n]",v.details.bike.type)
          getchar();
         v.fptr = display_bike;
         break;

      case 3:
```

```
            printf("Enter details of Truck:\n");
            printf("Enter the load capacity: ");
            scanf("%d", &v.details.truck.load_capacity);

            printf("Enter the number of axles: ");
            scanf("%d", &v.details.truck.no_of_axels);
            v.fptr = display_truck;
            break;

        default:
            printf("\nWrong choice\n");
    }
}

void Display_vehicle()
{
    printf("Display details:\n");
    printf("Manufacturer name: %s", v.manufacturer_name);
    printf("Model year: %d\n", v.model_year);

    printf("Vehicle type: ");
    if(v.vehicle_type == 1)
    {
        printf("Car\n");
    }
    else if(v.vehicle_type == 2)
    {
        printf("Bike\n");
    }
    else if(v.vehicle_type == 3)
    {
        printf("Truck\n");
    }
    else
    {
        printf("Unknown\n");
    }

    v.fptr(&v);
}

void display_car(struct VEHICLE *v)
```

```
{
    printf("Number of Doors: %d\n", v->details.car.no_of_doors);
    printf("Seating Capacity: %d\n", v->details.car.seats);
}

void display_bike(struct VEHICLE *v)
{
    printf("Engine Capacity: %d cc\n", v->details.bike.engine_capacity);
    printf("Bike Type: %s", v->details.bike.type);
}

void display_truck(struct VEHICLE *v)
{
    printf("Load Capacity: %d kg\n", v->details.truck.load_capacity);
    printf("Number of Axles: %d\n", v->details.truck.no_of_axels);
}
```

RECURSION

1.FACTORIAL

WAP to find out the factorial of a number using recursion.

```
#include<stdio.h>

int fact(int);
int main()
{
    int n;
    printf("Enter number:");
    scanf("%d",&n);

   int result=fact(n);
   printf("factorial %d",result);

}
int fact(int num)
{
    if(num==0  || num==1)
    return 1;
    else
```

```
    return num*fact(num-1);

}
```

2. WAP to find the sum of digits of a number using recursion.

```
#include<stdio.h>

int sum_of_digits(int);
int main()
{
    int n;
    printf("Enter number:");
    scanf("%d",&n);

   int result=sum_of_digits(n);
   printf("sum= %d",result);

}

int sum_of_digits(int num)
{
    if(num==0)
    return 0;

  return (num%10)+sum_of_digits(num/10);

}
```

3. With Recursion Findout the maximum number in a given array

```
 #include <stdio.h>

int find_max(int*,int);

int main()
{
    int arr[5] = {3, 1, 4, 1, 5};
    int n=5;
    printf("Maximum number in the array is: %d\n", find_max(arr, n));
    return 0;
}
int find_max(int arr[], int n)
```

```
{
    if (n == 1)
        return arr[0];

    int maxi = find_max(arr, n - 1);

    if(arr[n - 1] > maxi)
    return arr[n - 1];
    else
    return maxi;
}
```

4. With recurion calculate the power of a given number

```
#include <stdio.h>

int Power(int,int);

int main()

{

    int base,power;

    printf("Enter base and power:");

    scanf("%d %d",&base,&power);

    printf("%d raised to the power %d is: %d\n", base, power, Power(base,power));

    return 0;

}

int Power(int base, int power)

{

    if (power == 0)

        return 1;

    return base * Power(base, power - 1);

}
```

5. With Recursion calculate the length of a string.

```c
#include <stdio.h>

int string_length(char *);
int main()
{
   char str[] = "hello";
   printf("Length of the string '%s' is: %d\n", str, string_length(str));
   return 0;
}

int string_length(char *str)
{
   if (*str == '\0')
      return 0;

   return 1 + string_length(str + 1);
}
```

6. With recursion revrsal of a string

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

void reverse_string(char *str, int start, int end) ;

int main()
{
   char str[] = "hello";
   reverse_string(str, 0, strlen(str) - 1);
   printf("Reversed string is: '%s'\n", str);
   return 0;
}

void reverse_string(char *str, int start, int end)
{
   if (start >= end)
      return;

   char temp = str[start];
   str[start] = str[end];
   str[end] = temp;
   reverse_string(str, start + 1, end - 1);
}
```