```c
/*
Simulate the motion of particles in a two-dimensional space under the
influence of forces.
Specifications:
Structure: Represents particle properties (mass, position, velocity).
Array: Stores the position and velocity vectors of multiple particles.
Union: Handles force types (gravitational, electric, or magnetic).
Strings: Define force types applied to particles.
const Pointers: Protect particle properties.
Double Pointers: Dynamically allocate memory for the particle system.
*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Force
{
    float force;
};
struct Property
{
    float velocity;
    float mass;
    char pos[8];
    int type;//0.size 1.crack;
    union Force f;

};
void add(struct Property **c,int *count)
{
    printf("Enter Velocity : ");
    scanf("%f",&(*c)[*count].velocity);
    printf("Enter mass  : ");
    scanf("%f",&(*c)[*count].mass);
    printf("Enter position : ");
    scanf("%s",(*c)[*count].pos);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter gravitational force : ");
```

```
    scanf("%f",&(*c)[*count].f.force);
    }
    else if((*c)[*count].type==1)
    {
        printf("Enter electric force : ");
       scanf("%f",&(*c)[*count].f.force);
    }
    else
    {
        printf("Enter magnetic force : ");
       scanf("%f",&(*c)[*count].f.force);
    }
(*count)++;
}
void delete(struct Property **c,int *count,int mass)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if((*c)[i].mass==mass)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Property not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Property *c,int count)
{
    printf("\nProcess details \n");
```

```
printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Velocity : %.2f | Mass : %.2f | Position : %s
|",c[i].velocity,c[i].mass,c[i].pos);
        if(c[i].type==0)
        {
            printf("Force Type : Gravity | Force : %.2f\n",c[i].f.force);
        }
        else if(c[i].type==1)
        {
            printf("Force Type : Electric | Force : %.2f\n",c[i].f.force);
        }
        else
        {
            printf("Force Type : Magnetic | Force : %.2f\n",c[i].f.force);
        }

printf("------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Property *t=(struct Property *)malloc(10*sizeof(struct
Property));
    int choice,count=0;
    int mass;
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
```

```
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter mass to remmove : ");
            scanf("%d",mass);
            delete(&t,&count,mass);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Calculate the electromagnetic field intensity at various points in
space.
Specifications:
Structure: Stores field Forces (electric field, magnetic field, and
position).
Array: Holds field values at discrete points.
Union: Represents either electric or magnetic field components.
Strings: Represent coordinate systems (Cartesian, cylindrical, spherical).
const Pointers: Prevent modification of field Forces.
Double Pointers: Manage memory for field grid allocation dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Force
{
    float electric;
    float magnetic;
};
struct Field
{
    char id[10];
    char pos[10];
    int sys ;//0Cartesian, 1 cylindrical, 2 spherical
```

```
    int type;//0.electric 1.magnetic;
    union Force f;

};
void add(struct Field **c,int *count)
{
    printf("Enter element id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter pos  : ");
    scanf("%s",(*c)[*count].pos);
    printf("Enter sys : ");
    scanf("%d",&(*c)[*count].sys);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter electric : ");
    scanf("%f",&(*c)[*count].f.electric);
    }
    else
    {
        printf("Enter magnetic : ");
       scanf("%f",&(*c)[*count].f.magnetic);
    }
(*count)++;
}
void delete(struct Field **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
```

```
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Field *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Element id : %s | pos : %s |",c[i].id,c[i].pos);
        if(c[i].sys==0)
        {
            printf("System : Cartesian |");
        }
        else if(c[i].sys==1)
        {
            printf("System : Cylindrical |");
        }
        else
        {
            printf("System : Spherical |");
        }
        if(c[i].type==0)
        {
            printf("Electric force : %.2f\n",c[i].f.electric);
        }
        else
        {
            printf("Magnetic force : %.2f\n",c[i].f.magnetic);
        }

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

```
    }
}
void main()
{
    struct Field *t=(struct Field *)malloc(10*sizeof(struct Field));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Track the energy levels of atoms and the transitions between them.
Specifications:
Structure: Contains atomic details (element name, energy levels, and
transition probabilities).
Array: Stores energy levels for different atoms.
Union: Represents different energy states.
```

```c
/*Strings: Represent element names.
const Pointers: Protect atomic data.
Double Pointers: Allocate memory for dynamically adding new elements.
*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Energy
{
    float p;
    float n;
};
struct Atom
{
    char id[10];
    char name[10];
    float lvl;
    float prb;
    int type;//0.p 1.n;
    union Energy e;

};
void add(struct Atom **c,int *count)
{
    printf("Enter element id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter name  : ");
    scanf("%s",(*c)[*count].name);
    printf("Enter Energy lvl : ");
    scanf("%f",&(*c)[*count].lvl);
    printf("Enter transition probability : ");
    scanf("%f",&(*c)[*count].prb);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter p : ");
    scanf("%f",&(*c)[*count].e.p);
    }
    else
```

```
    {
        printf("Enter n : ");
       scanf("%f",&(*c)[*count].e.n);
    }
(*count)++;
}
void delete(struct Atom **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Atom *c,int count)
{
    printf("\nProcess details \n");

printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Element id : %s | Name : %s | Energy level : %.2f |
Probability : %.2f | ",c[i].id,c[i].name,c[i].lvl,c[i].prb);
        if(c[i].type==0)
        {
```

```
            printf("P energy : %.2f\n",c[i].e.p);
        }
        else
        {
            printf("N Energy : %.2f\n",c[i].e.n);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Atom *t=(struct Atom *)malloc(10*sizeof(struct Atom));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
```

```c
    } while (choice !=4);

}
/*Develop a program to represent quantum states and their evolution over
time.
Specifications:
Structure: Holds state properties (wavefunction amplitude, phase, and
energy).
Array: Represents the wavefunction across multiple points.
Union: Stores amplitude or phase information.
Strings: Describe state labels (e.g., "ground state," "excited state").
const Pointers: Protect state properties.
Double Pointers: Manage quantum states dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Energy
{
    float amplitude;
    float phase;
};
struct State
{
    char id[10];
    char state[10];
    float energy;
    int type;//0.amplitude 1.phase;
    union Energy e;

};
void add(struct State **c,int *count)
{
    printf("Enter state id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter state  : ");
    scanf("%s",(*c)[*count].state);
    printf("Enter Energy : ");
    scanf("%f",&(*c)[*count].energy);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
```

```
    if((*c)[*count].type==0)
    {
        printf("Enter amplitude : ");
    scanf("%f",&(*c)[*count].e.amplitude);
    }
    else
    {
        printf("Enter phase : ");
       scanf("%f",&(*c)[*count].e.phase);
    }
(*count)++;
}
void delete(struct State **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct State *c,int count)
{
    printf("\nProcess details \n");

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

```
    for(int i=0;i<count;i++)
    {
        printf("State id : %s | State : %s | Energy : %.2f |
",c[i].id,c[i].state,c[i].energy);
        if(c[i].type==0)
        {
            printf("Amplitude : %.2f\n",c[i].e.amplitude);
        }
        else
        {
            printf("Phase : %.2f\n",c[i].e.phase);
        }

printf("-------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct State *t=(struct State *)malloc(10*sizeof(struct State));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
```

```
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Simulate light rays passing through different optical elements.
Specifications:
Structure: Represents optical properties (refractive index, focal length).
Array: Stores light ray paths.
Union: Handles lens or mirror parameters.
Strings: Represent optical element types.
const Pointers: Protect optical properties.
Double Pointers: Manage arrays of optical elements dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float lens;
    float mirror;
};
struct Lens
{
    char id[10];
    float ri;
    float fl;
    int type;//0.lens 1.mirror;
    union Parameter e;

};
void add(struct Lens **c,int *count)
{
    printf("Enter Lens id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter refractive index  : ");
    scanf("%f",&(*c)[*count].ri);
```

```
    printf("Enter focal length : ");
    scanf("%f",&(*c)[*count].fl);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter lens parameter : ");
    scanf("%f",&(*c)[*count].e.lens);
    }
    else
    {
        printf("Enter mirror parameter : ");
       scanf("%f",&(*c)[*count].e.mirror);
    }
(*count)++;
}
void delete(struct Lens **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Lens *c,int count)
{
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Lens id : %s | Refractive index : %.2f | Focal length :
%.2f | ",c[i].id,c[i].ri,c[i].fl);
        if(c[i].type==0)
        {
            printf("Lens parametrs : %.2f\n",c[i].e.lens);
        }
        else
        {
            printf("mirror parameters : %.2f\n",c[i].e.mirror);
        }

printf("------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Lens *t=(struct Lens *)malloc(10*sizeof(struct Lens));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
```

```
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Calculate thermodynamic states of a system based on input parameters
like pressure, volume, and temperature.
Specifications:
Structure: Represents thermodynamic properties (P, V, T, and entropy).
Array: Stores states over a range of conditions.
Union: Handles dependent properties like energy or entropy.
Strings: Represent state descriptions.
const Pointers: Protect thermodynamic data.
Double Pointers: Allocate state data dynamically for simulation.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float energy;
    float entropy;
};
struct Thermal
{
    char id[10];
    float pressure;
    float volume;
    float temp;
    int type;//0.energy 1.entropy;
    union Parameter e;

};
```

```c
void add(struct Thermal **c,int *count)
{
    printf("Enter Thermal id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter pressure  : ");
    scanf("%f",&(*c)[*count].pressure);
    printf("Enter focal volume : ");
    scanf("%f",&(*c)[*count].volume);
    printf("Enter temperature : ");
    scanf("%f",&(*c)[*count].temp);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter energy : ");
    scanf("%f",&(*c)[*count].e.energy);
    }
    else
    {
        printf("Enter entropy : ");
       scanf("%f",&(*c)[*count].e.entropy);
    }
(*count)++;
}
void delete(struct Thermal **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
```

```
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Thermal *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Thermal id : %s | Pressure : %.2f | Volume : %.2f |
Temperature : %.2f ",c[i].id,c[i].pressure,c[i].volume,c[i].temp);
        if(c[i].type==0)
        {
            printf("Energy : %.2f\n",c[i].e.energy);
        }
        else
        {
            printf("Entropy : %.2f\n",c[i].e.entropy);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Thermal *t=(struct Thermal *)malloc(10*sizeof(struct Thermal));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
```

```
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Track the parameters of nuclear reactions like fission and fusion
processes.
Specifications:
Structure: Represents reaction details (reactants, products, energy
released).
Array: Holds data for multiple reactions.
Union: Represents either energy release or product details.
Strings: Represent reactant and product names.
const Pointers: Protect reaction details.
Double Pointers: Dynamically allocate memory for reaction data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float energy;
    char product[20];
};
struct Reaction
```

```
{
    char id[10];
    char reactants[20];
    int type;//0.energy 1.product
    union Parameter e;

};
void add(struct Reaction **c,int *count)
{
    printf("Enter Reaction id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter reactants  : ");
    getchar();
    fgets((*c)[*count].reactants,20,stdin);
    (*c)[*count].reactants[strcspn((*c)[*count].reactants,"\n")]='0';
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter energy : ");
    scanf("%f",&(*c)[*count].e.energy);
    }
    else
    {
        printf("Enter product : ");
       scanf("%s",(*c)[*count].e.product);
    }
(*count)++;
}
void delete(struct Reaction **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
```

```
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Reaction *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Reaction id : %s | Reactants : %.s |
",c[i].id,c[i].reactants);
        if(c[i].type==0)
        {
            printf("Energy : %.2f\n",c[i].e.energy);
        }
        else
        {
            printf("product : %s\n",c[i].e.product);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Reaction *t=(struct Reaction *)malloc(10*sizeof(struct
Reaction));
    int choice,count=0;
    char id[10];
```

```
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Simulate the gravitational field of massive objects in a system.
Specifications:
Structure: Contains object properties (mass, position, field strength).
Array: Stores field values at different points.
Union: Handles either mass or field strength as parameters.
Strings: Represent object labels (e.g., "Planet A," "Star B").
const Pointers: Protect object properties.
Double Pointers: Dynamically allocate memory for gravitational field
data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
```

```c
{
    float mass;
    float strength;
};
struct Planet
{
    char id[10];
    char label[20];
    char position[20];
    int type;//0.mass 1.strength
    union Parameter e;

};
void add(struct Planet **c,int *count)
{
    printf("Enter Planet id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter label :");
    scanf("%s",(*c)[*count].label);
    printf("Enter position  : ");
    getchar();
    fgets((*c)[*count].position,20,stdin);
    (*c)[*count].position[strcspn((*c)[*count].position,"\n")]='0';
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter mass : ");
    scanf("%f",&(*c)[*count].e.mass);
    }
    else
    {
        printf("Enter strength : ");
       scanf("%f",&(*c)[*count].e.strength);
    }
(*count)++;
}
void delete(struct Planet **c,int *count,const char *id)
{
    int index=-1;
```

```
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Planet *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Planet id : %s |Label : %s | position : %s |
",c[i].id,c[i].label,c[i].position);
        if(c[i].type==0)
        {
            printf("Mass : %.2f\n",c[i].e.mass);
        }
        else
        {
            printf("Strength : %.2f\n",c[i].e.strength);
        }

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

```
    }
}
void main()
{
    struct Planet *t=(struct Planet *)malloc(10*sizeof(struct Planet));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Analyze interference patterns produced by waves from multiple sources.
Specifications:
Structure: Represents wave properties (amplitude, wavelength, and phase).
Array: Stores wave interference data at discrete points.
Union: Handles either amplitude or phase information.
Strings: Represent wave source labels.
```

```c
const Pointers: Protect wave properties.
Double Pointers: Manage dynamic allocation of wave sources.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float amplitude;
    float phase;
};
struct Wave
{
    char id[10];
    char label[20];
    float wavelength;
    int type;//0.amplitude 1.phase
    union Parameter e;

};
void add(struct Wave **c,int *count)
{
    printf("Enter Wave id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter label :");
    scanf("%s",(*c)[*count].label);
    printf("Enter wavelength  : ");
    scanf("%f",&(*c)[*count].wavelength);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter amplitude : ");
    scanf("%f",&(*c)[*count].e.amplitude);
    }
    else
    {
        printf("Enter phase : ");
       scanf("%f",&(*c)[*count].e.phase);
    }
(*count)++;
```

```
}
void delete(struct Wave **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Wave *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Wave id : %s |Label : %s | Wavelength : %.2f |
",c[i].id,c[i].label,c[i].wavelength);
        if(c[i].type==0)
        {
            printf("Amplitude : %.2f\n",c[i].e.amplitude);
        }
        else
        {
            printf("Phase : %.2f\n",c[i].e.phase);
```

```
        }

printf("-------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Wave *t=(struct Wave *)malloc(10*sizeof(struct Wave));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Create a database to store and retrieve properties of magnetic
materials.
```

```c
/*Specifications:
Structure: Represents material properties (permeability, saturation).
Array: Stores data for multiple materials.
Union: Handles temperature-dependent properties.
Strings: Represent material names.
const Pointers: Protect material data.
Double Pointers: Allocate material records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float expansion;
    float shrinkage;
};
struct Material
{
    char id[10];
    char name[20];
    float saturation;
    int type;//0.expansion 1.shrinkage
    union Parameter e;

};
void add(struct Material **c,int *count)
{
    printf("Enter Material id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter name :");
    scanf("%s",(*c)[*count].name);
    printf("Enter saturation  : ");
    scanf("%f",&(*c)[*count].saturation);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter expansion : ");
    scanf("%f",&(*c)[*count].e.expansion);
    }
    else
```

```
    {
        printf("Enter shrinkage : ");
       scanf("%f",&(*c)[*count].e.shrinkage);
    }
(*count)++;
}
void delete(struct Material **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Material *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Material id : %s |Name : %s | Saturation : %.2f |
",c[i].id,c[i].name,c[i].saturation);
        if(c[i].type==0)
        {
```

```
            printf("Expansion : %.2f\n",c[i].e.expansion);
        }
        else
        {
            printf("Shrinkage : %.2f\n",c[i].e.shrinkage);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Material *t=(struct Material *)malloc(10*sizeof(struct
Material));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
```

```
        }
    } while (choice !=4);

}
/*Simulate the behavior of plasma under various conditions.
Specifications:
Structure: Represents plasma parameters (density, temperature, and
electric field).
Array: Stores simulation results.
Union: Handles either density or temperature data.
Strings: Represent plasma types.
const Pointers: Protect plasma parameters.
Double Pointers: Manage dynamic allocation for simulation data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float density;
    float temperature;
};
struct Plasma
{
    char id[10];
    char name[20];
    float field;
    int type;//0.density 1.temperature
    union Parameter e;

};
void add(struct Plasma **c,int *count)
{
    printf("Enter Plasma id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter plasma type :");
    scanf("%s",(*c)[*count].name);
    printf("Enter electric field  : ");
    scanf("%f",&(*c)[*count].field);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
```

```
    if((*c)[*count].type==0)
    {
        printf("Enter density : ");
    scanf("%f",&(*c)[*count].e.density);
    }
    else
    {
        printf("Enter temperature : ");
       scanf("%f",&(*c)[*count].e.temperature);
    }
(*count)++;
}
void delete(struct Plasma **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Plasma *c,int count)
{
    printf("\nProcess details \n");

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

```
    for(int i=0;i<count;i++)
    {
        printf("Plasma id : %s |Plasma type: %s | Electric field : %.2f |
",c[i].id,c[i].name,c[i].field);
        if(c[i].type==0)
        {
            printf("Density : %.2f\n",c[i].e.density);
        }
        else
        {
            printf("Temperature : %.2f\n",c[i].e.temperature);
        }

printf("-------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Plasma *t=(struct Plasma *)malloc(10*sizeof(struct Plasma));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
```

```
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Solve complex kinematics problems for objects in motion.
Specifications:
Structure: Represents object properties (initial velocity, acceleration,
displacement).
Array: Stores time-dependent motion data.
Union: Handles either velocity or displacement equations.
Strings: Represent motion descriptions.
const Pointers: Protect object properties.
Double Pointers: Dynamically allocate memory for motion data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float velocity;
    float displacement;
};
struct Object
{
    char id[10];
    char description[20];
    float acceleration;
    int type;//0.velocity 1.displacement
    union Parameter e;

};
void add(struct Object **c,int *count)
{
    printf("Enter Object id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter motion description :");
```

```
    getchar();
    fgets((*c)[*count].description,20,stdin);
    (*c)[*count].description[strcspn((*c)[*count].description,"\n")]='\0';
    printf("Enter acceleration  : ");
    scanf("%f",&(*c)[*count].acceleration);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter velocity : ");
    scanf("%f",&(*c)[*count].e.velocity);
    }
    else
    {
        printf("Enter displacement : ");
       scanf("%f",&(*c)[*count].e.displacement);
    }
(*count)++;
}
void delete(struct Object **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
```

```
}
void display(struct Object *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Object id : %s |Motion description : %s | Acceleration :
%.2f | ",c[i].id,c[i].description,c[i].acceleration);
        if(c[i].type==0)
        {
            printf("velocity : %.2f\n",c[i].e.velocity);
        }
        else
        {
            printf("displacement : %.2f\n",c[i].e.displacement);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Object *t=(struct Object *)malloc(10*sizeof(struct Object));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
```

```
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Develop a database to store and analyze spectral lines of elements.
Specifications:
Structure: Represents line properties (wavelength, intensity, and
element).
Array: Stores spectral line data.
Union: Handles either intensity or wavelength information.
Strings: Represent element names.
const Pointers: Protect spectral line data.
Double Pointers: Allocate spectral line records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Paelementter
{
    float intensity;
    float wavelength;
};
struct Line
{
    char id[10];
    char element[20];
    int type;//0.intensity 1.wavelength
    union Paelementter e;
```

```
};
void add(struct Line **c,int *count)
{
    printf("Enter Line id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter element name:");
    getchar();
    fgets((*c)[*count].element,20,stdin);
    (*c)[*count].element[strcspn((*c)[*count].element,"\n")]='\0';
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter intensity : ");
    scanf("%f",&(*c)[*count].e.intensity);
    }
    else
    {
        printf("Enter wavelength : ");
       scanf("%f",&(*c)[*count].e.wavelength);
    }
(*count)++;
}
void delete(struct Line **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Element not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
```

```
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Line *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Line id : %s | Element name : %s |
",c[i].id,c[i].element);
        if(c[i].type==0)
        {
            printf("Intensity : %.2f\n",c[i].e.intensity);
        }
        else
        {
            printf("Wavelength : %.2f\n",c[i].e.wavelength);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Line *t=(struct Line *)malloc(10*sizeof(struct Line));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Simulate and analyze projectile motion under varying conditions.
Specifications:
Structure: Stores projectile properties (mass, velocity, and angle).
Array: Stores motion trajectory data.
Union: Handles either velocity or displacement parameters.
Strings: Represent trajectory descriptions.
const Pointers: Protect projectile properties.
Double Pointers: Manage trajectory records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parametr
{
    float velocity;
    float displacement;
};
struct Projectile
{
    char id[10];
    char desc[20];
```

```
    float mass;
    float angle;
    int type;//0.velocity 1.displacement
    union Parametr e;

};
void add(struct Projectile **c,int *count)
{
    printf("Enter Projectile id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter projectile trajectory description :");
    getchar();
    fgets((*c)[*count].desc,20,stdin);
    (*c)[*count].desc[strcspn((*c)[*count].desc,"\n")]='\0';
    printf("Enter mass : ");
    scanf("%f",&(*c)[*count].mass);
    printf("Enter angle : ");
    scanf("%f",&(*c)[*count].angle);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter velocity : ");
    scanf("%f",&(*c)[*count].e.velocity);
    }
    else
    {
        printf("Enter displacement : ");
       scanf("%f",&(*c)[*count].e.displacement);
    }
(*count)++;
}
void delete(struct Projectile **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
```

```
            break;
        }
    }
    if(index==-1)
    {
        printf("desc not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Projectile *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Projectile id : %s |  Description : %s | Mass : %.2f |
Angle : %.2f ",c[i].id,c[i].desc,c[i].mass,c[i].angle);
        if(c[i].type==0)
        {
            printf("Velocity : %.2f\n",c[i].e.velocity);
        }
        else
        {
            printf("Displacement : %.2f\n",c[i].e.displacement);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
```

```
    struct Projectile *t=(struct Projectile *)malloc(10*sizeof(struct
Projectile));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
/*Analyze the stress-strain behavior of materials under different loads.
Specifications:
Structure: Represents material properties (stress, strain, modulus).
Array: Stores stress-strain data.
Union: Handles dependent properties like yield stress or elastic modulus.
Strings: Represent material names.
const Pointers: Protect material properties.
Double Pointers: Allocate stress-strain data dynamically.*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
union Parametr
{
    float yeilds;
    float modulus;
};
struct Material
{
    char id[10];
    char name[20];
    float stress;
    float strain;
    int type;//0.yeild stress 1.elastic modulus
    union Parametr e;

};
void add(struct Material **c,int *count)
{
    printf("Enter Material id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material name :");
    getchar();
    fgets((*c)[*count].name,20,stdin);
    (*c)[*count].name[strcspn((*c)[*count].name,"\n")]='\0';
    printf("Enter stress : ");
    scanf("%f",&(*c)[*count].stress);
    printf("Enter strain : ");
    scanf("%f",&(*c)[*count].strain);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter yeilds : ");
    scanf("%f",&(*c)[*count].e.yeilds);
    }
    else
    {
     (*c)[*count].e.modulus=(*c)[*count].stress/(*c)[*count].strain;
    }
```

```
(*count)++;
}
void delete(struct Material **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("name not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Material *c,int count)
{
    printf("\nProcess details \n");

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Material id : %s | Name : %s | Stress : %.2f | Strain :
%.2f | ",c[i].id,c[i].name,c[i].stress,c[i].strain);
        if(c[i].type==0)
        {
            printf("Stress yeilds : %.2f\n",c[i].e.yeilds);
        }
        else
        {
```

```
            printf("Elastic modulus : %.2f\n",c[i].e.modulus);
        }

printf("------------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Material *t=(struct Material *)malloc(10*sizeof(struct
Material));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
            break;
        }
    } while (choice !=4);

}
```