```
//1.
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define GRAVITY 9.8

typedef struct {
    double x, y;
} Vector2D;

typedef struct {
    double mass;
    Vector2D position;
    Vector2D velocity;
} Particle;

typedef union {
    double gravitationalForce;
    double electricForce;
    double magneticForce;
} ForceType;

typedef enum {
    GRAVITATIONAL,
    ELECTRIC,
    MAGNETIC
} Force;

void applyForce(Particle *p, ForceType force, Force type) {
    switch (type) {
```

```
        case GRAVITATIONAL:
            p->velocity.y -= force.gravitationalForce / p->mass;
            break;
        case ELECTRIC:
            p->velocity.x += force.electricForce / p->mass;
            break;
        case MAGNETIC:
            p->velocity.y -= force.magneticForce / p->mass;
            break;
        default:
            printf("Unknown force type!\n");
            break;
    }
}

void updateParticle(Particle *p, double deltaTime) {
    p->position.x += p->velocity.x * deltaTime;
    p->position.y += p->velocity.y * deltaTime;
}

void printParticleState(Particle *p) {
    printf("Position: (%.2f, %.2f) Velocity: (%.2f, %.2f)\n",
           p->position.x, p->position.y, p->velocity.x, p->velocity.y);
}

int main() {
    int numParticles = 3;
    Particle **particleSystem = (Particle **)malloc(numParticles * sizeof(Particle *));
    if (particleSystem == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
```

```
}

for (int i = 0; i < numParticles; i++) {
   particleSystem[i] = (Particle *)malloc(sizeof(Particle));
   if (particleSystem[i] == NULL) {
      printf("Memory allocation failed!\n");
      return 1;
   }
   particleSystem[i]->mass = 1.0;
   particleSystem[i]->position.x = 0.0;
   particleSystem[i]->position.y = 0.0;
   particleSystem[i]->velocity.x = 0.0;
   particleSystem[i]->velocity.y = 0.0;
}

ForceType force;
force.gravitationalForce = GRAVITY;

double deltaTime = 0.1;
for (int t = 0; t <= 5; t++) { // Time steps from 0 to 5
   printf("\nTime step %d:\n", t);
   for (int i = 0; i < numParticles; i++) {
      applyForce(particleSystem[i], force, GRAVITATIONAL);
      updateParticle(particleSystem[i], deltaTime);
      printParticleState(particleSystem[i]);
   }
}

for (int i = 0; i < numParticles; i++) {
   free(particleSystem[i]);
}
```

```
    free(particleSystem);

    return 0;
}

//2.

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

typedef struct {
    double Ex, Ey, Ez;
    double Bx, By, Bz;
    double x, y, z;
} FieldPoint;

typedef union {
    double electricField[3];
    double magneticField[3];
} FieldComponents;

typedef enum {
    CARTESIAN,
    CYLINDRICAL,
    SPHERICAL
} CoordinateSystem;

void printField(FieldPoint *point, CoordinateSystem coordSys) {
    printf("Position: (%.2f, %.2f, %.2f)\n", point->x, point->y, point->z);
    if (coordSys == CARTESIAN) {
```

```
        printf("Electric Field: (%.2f, %.2f, %.2f)\n", point->Ex, point->Ey, point->Ez);
        printf("Magnetic Field: (%.2f, %.2f, %.2f)\n", point->Bx, point->By, point->Bz);
    } else if (coordSys == CYLINDRICAL) {
    } else if (coordSys == SPHERICAL) {
    }
}

int main() {
    int numPoints = 3;
    FieldPoint **fieldGrid = (FieldPoint **)malloc(numPoints * sizeof(FieldPoint *));
    if (fieldGrid == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numPoints; i++) {
        fieldGrid[i] = (FieldPoint *)malloc(sizeof(FieldPoint));
        if (fieldGrid[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        fieldGrid[i]->x = i * 1.0;
        fieldGrid[i]->y = i * 1.0;
        fieldGrid[i]->z = i * 1.0;
        fieldGrid[i]->Ex = 0.5 * i;
        fieldGrid[i]->Ey = 0.5 * i;
        fieldGrid[i]->Ez = 0.5 * i;
        fieldGrid[i]->Bx = 0.2 * i;
        fieldGrid[i]->By = 0.2 * i;
        fieldGrid[i]->Bz = 0.2 * i;
```

```
    }

    CoordinateSystem coordSys = CARTESIAN;
    for (int i = 0; i < numPoints; i++) {
        printField(fieldGrid[i], coordSys);
    }

    for (int i = 0; i < numPoints; i++) {
        free(fieldGrid[i]);
    }
    free(fieldGrid);

    return 0;
}

//3.

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

typedef struct {
    double Ex, Ey, Ez;
    double Bx, By, Bz;
    double x, y, z;
} FieldPoint;

typedef union {
    double electricField[3];
    double magneticField[3];
} FieldComponents;
```

```
typedef enum {
    CARTESIAN,
    CYLINDRICAL,
    SPHERICAL
} CoordinateSystem;

void printField(FieldPoint *point, CoordinateSystem coordSys) {
    printf("Position: (%.2f, %.2f, %.2f)\n", point->x, point->y, point->z);
    if (coordSys == CARTESIAN) {
        printf("Electric Field: (%.2f, %.2f, %.2f)\n", point->Ex, point->Ey, point->Ez);
        printf("Magnetic Field: (%.2f, %.2f, %.2f)\n", point->Bx, point->By, point->Bz);
    } else if (coordSys == CYLINDRICAL) {
    } else if (coordSys == SPHERICAL) {
    }
}

int main() {
    int numPoints = 3;
    FieldPoint **fieldGrid = (FieldPoint **)malloc(numPoints * sizeof(FieldPoint *));
    if (fieldGrid == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numPoints; i++) {
        fieldGrid[i] = (FieldPoint *)malloc(sizeof(FieldPoint));
        if (fieldGrid[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }
```

```
        fieldGrid[i]->x = i * 1.0;
        fieldGrid[i]->y = i * 1.0;
        fieldGrid[i]->z = i * 1.0;
        fieldGrid[i]->Ex = 0.5 * i;
        fieldGrid[i]->Ey = 0.5 * i;
        fieldGrid[i]->Ez = 0.5 * i;
        fieldGrid[i]->Bx = 0.2 * i;
        fieldGrid[i]->By = 0.2 * i;
        fieldGrid[i]->Bz = 0.2 * i;
    }

    CoordinateSystem coordSys = CARTESIAN;
    for (int i = 0; i < numPoints; i++) {
        printField(fieldGrid[i], coordSys);
    }

    for (int i = 0; i < numPoints; i++) {
        free(fieldGrid[i]);
    }
    free(fieldGrid);

    return 0;
}

//4.
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct {
```

```
    double amplitude;
    double phase;
    double energy;
} QuantumState;

typedef union {
    double amplitude;
    double phase;
} StateProperty;

typedef enum {
    GROUND_STATE,
    EXCITED_STATE
} StateType;

void printQuantumState(QuantumState *state, StateType type) {
    printf("State Type: %s\n", type == GROUND_STATE ? "Ground State" : "Excited State");
    printf("Amplitude: %.2f\n", state->amplitude);
    printf("Phase: %.2f rad\n", state->phase);
    printf("Energy: %.2f eV\n", state->energy);
}

int main() {
    int numStates = 3;
    QuantumState **quantumStates = (QuantumState **)malloc(numStates * sizeof(QuantumState
*));
    if (quantumStates == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
```

```
    for (int i = 0; i < numStates; i++) {
        quantumStates[i] = (QuantumState *)malloc(sizeof(QuantumState));
        if (quantumStates[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        quantumStates[i]->amplitude = (i + 1) * 0.5;
        quantumStates[i]->phase = (i + 1) * 0.2;
        quantumStates[i]->energy = (i + 1) * 1.0;
    }

    StateType type = GROUND_STATE;
    for (int i = 0; i < numStates; i++) {
        printQuantumState(quantumStates[i], type);
        type = EXCITED_STATE;
    }

    for (int i = 0; i < numStates; i++) {
        free(quantumStates[i]);
    }
    free(quantumStates);

    return 0;
}

//5.

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

```
typedef struct {
    double refractiveIndex;
    double focalLength;
} OpticalElement;

typedef union {
    double lensParameters[2];
    double mirrorParameters[1];
} ElementParameters;

typedef enum {
    LENS,
    MIRROR
} ElementType;

void printOpticalElement(OpticalElement *element, ElementType type) {
    if (type == LENS) {
        printf("Element Type: Lens\n");
        printf("Refractive Index: %.2f\n", element->refractiveIndex);
        printf("Focal Length: %.2f\n", element->focalLength);
    } else if (type == MIRROR) {
        printf("Element Type: Mirror\n");
        printf("Refractive Index: %.2f\n", element->refractiveIndex);
    }
}

int main() {
    int numElements = 2;
    OpticalElement **opticalElements = (OpticalElement **)malloc(numElements *
sizeof(OpticalElement *));
    if (opticalElements == NULL) {
```

```
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numElements; i++) {
        opticalElements[i] = (OpticalElement *)malloc(sizeof(OpticalElement));
        if (opticalElements[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        opticalElements[i]->refractiveIndex = 1.5;
        opticalElements[i]->focalLength = 5.0;
    }

    ElementType type = LENS;
    for (int i = 0; i < numElements; i++) {
        printOpticalElement(opticalElements[i], type);
        type = MIRROR;
    }

    for (int i = 0; i < numElements; i++) {
        free(opticalElements[i]);
    }
    free(opticalElements);

    return 0;
}

//6.
```

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

typedef struct {
    double pressure;
    double volume;
    double temperature;
    double entropy;
} ThermodynamicState;

typedef union {
    double energy;
    double entropy;
} StateProperties;

typedef enum {
    GAS,
    LIQUID
} StateType;

void printThermodynamicState(ThermodynamicState *state, StateType type) {
    printf("State Type: %s\n", type == GAS ? "Gas" : "Liquid");
    printf("Pressure: %.2f Pa\n", state->pressure);
    printf("Volume: %.2f m^3\n", state->volume);
    printf("Temperature: %.2f K\n", state->temperature);
    printf("Entropy: %.2f J/K\n", state->entropy);
}

int main() {
    int numStates = 3;
```

```
    ThermodynamicState **states = (ThermodynamicState **)malloc(numStates *
sizeof(ThermodynamicState *));
    if (states == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numStates; i++) {
        states[i] = (ThermodynamicState *)malloc(sizeof(ThermodynamicState));
        if (states[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        states[i]->pressure = 1.0e5;
        states[i]->volume = 1.0;
        states[i]->temperature = 300.0;
        states[i]->entropy = 100.0;
    }

    StateType type = GAS;
    for (int i = 0; i < numStates; i++) {
        printThermodynamicState(states[i], type);
        type = LIQUID;
    }

    for (int i = 0; i < numStates; i++) {
        free(states[i]);
    }
    free(states);
```

```
    return 0;
}

//7.

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

typedef struct {
    char reactant[50];
    char product[50];
    double energyReleased;
} NuclearReaction;

typedef union {
    double energyReleased;
    char product[50];
} ReactionDetails;

void printReactionDetails(NuclearReaction *reaction) {
    printf("Reactant: %s\n", reaction->reactant);
    printf("Product: %s\n", reaction->product);
    printf("Energy Released: %.2f MeV\n", reaction->energyReleased);
}

int main() {
    int numReactions = 2;
    NuclearReaction **reactions = (NuclearReaction **)malloc(numReactions *
sizeof(NuclearReaction *));
    if (reactions == NULL) {
```

```
    printf("Memory allocation failed!\n");
    return 1;
}

for (int i = 0; i < numReactions; i++) {
    reactions[i] = (NuclearReaction *)malloc(sizeof(NuclearReaction));
    if (reactions[i] == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    if (i == 0) {
        strcpy(reactions[i]->reactant, "Uranium-235");
        strcpy(reactions[i]->product, "Krypton + Barium");
        reactions[i]->energyReleased = 200.0;
    } else {
        strcpy(reactions[i]->reactant, "Deuterium");
        strcpy(reactions[i]->product, "Helium");
        reactions[i]->energyReleased = 17.6;
    }
}

for (int i = 0; i < numReactions; i++) {
    printReactionDetails(reactions[i]);
}

for (int i = 0; i < numReactions; i++) {
    free(reactions[i]);
}
free(reactions);
```

```
    return 0;
}

//8.

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

#define G 6.67430e-11

typedef struct {
    double mass;
    double x, y, z;
    double fieldStrength;
} GravitationalObject;

typedef union {
    double mass;
    double fieldStrength;
} FieldParameters;

void calculateFieldStrength(GravitationalObject *object) {
    double r = sqrt(object->x * object->x + object->y * object->y + object->z * object->z);
    object->fieldStrength = G * object->mass / (r * r);
}

void printGravitationalObject(GravitationalObject *object, const char *label) {
    printf("Object: %s\n", label);
    printf("Mass: %.2e kg\n", object->mass);
    printf("Position: (%.2f, %.2f, %.2f)\n", object->x, object->y, object->z);
```

```
    printf("Gravitational Field Strength: %.2e N/kg\n", object->fieldStrength);
}

int main() {
    int numObjects = 2;
    GravitationalObject **objects = (GravitationalObject **)malloc(numObjects *
sizeof(GravitationalObject *));
    if (objects == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numObjects; i++) {
        objects[i] = (GravitationalObject *)malloc(sizeof(GravitationalObject));
        if (objects[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            objects[i]->mass = 5.97e24; // Mass of Earth
            objects[i]->x = 0;
            objects[i]->y = 0;
            objects[i]->z = 0;
        } else {
            objects[i]->mass = 1.99e30; // Mass of Sun
            objects[i]->x = 1.496e11; // Distance from Earth (1 AU)
            objects[i]->y = 0;
            objects[i]->z = 0;
        }
```

```
        calculateFieldStrength(objects[i]);
    }

    const char *labels[] = {"Earth", "Sun"};
    for (int i = 0; i < numObjects; i++) {
        printGravitationalObject(objects[i], labels[i]);
    }

    for (int i = 0; i < numObjects; i++) {
        free(objects[i]);
    }
    free(objects);

    return 0;
}

//9.

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

typedef struct {
    double amplitude;
    double wavelength;
    double phase;
} Wave;

typedef union {
    double amplitude;
    double phase;
```

```
} WaveProperty;

void printWaveProperties(Wave *wave, const char *label) {
    printf("Wave Source: %s\n", label);
    printf("Amplitude: %.2f\n", wave->amplitude);
    printf("Wavelength: %.2f m\n", wave->wavelength);
    printf("Phase: %.2f rad\n", wave->phase);
}

int main() {
    int numWaves = 2;
    Wave **waves = (Wave **)malloc(numWaves * sizeof(Wave *));
    if (waves == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numWaves; i++) {
        waves[i] = (Wave *)malloc(sizeof(Wave));
        if (waves[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            waves[i]->amplitude = 1.0;
            waves[i]->wavelength = 500.0;
            waves[i]->phase = 0.0;
        } else {
            waves[i]->amplitude = 0.5;
            waves[i]->wavelength = 600.0;
```

```
            waves[i]->phase = M_PI / 2;
        }
    }

    const char *labels[] = {"Wave 1", "Wave 2"};
    for (int i = 0; i < numWaves; i++) {
        printWaveProperties(waves[i], labels[i]);
    }

    for (int i = 0; i < numWaves; i++) {
        free(waves[i]);
    }
    free(waves);

    return 0;
}

//10.

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

typedef struct {
    char materialName[50];
    double permeability;
    double saturation;
} MagneticMaterial;

typedef union {
    double permeability;
```

```
    double saturation;
} MaterialProperty;

void printMaterialProperties(MagneticMaterial *material) {
    printf("Material: %s\n", material->materialName);
    printf("Permeability: %.2e H/m\n", material->permeability);
    printf("Saturation: %.2e A/m\n", material->saturation);
}

int main() {
    int numMaterials = 2;
    MagneticMaterial **materials = (MagneticMaterial **)malloc(numMaterials *
sizeof(MagneticMaterial *));
    if (materials == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numMaterials; i++) {
        materials[i] = (MagneticMaterial *)malloc(sizeof(MagneticMaterial));
        if (materials[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            strcpy(materials[i]->materialName, "Iron");
            materials[i]->permeability = 1.26e-6;
            materials[i]->saturation = 2.2e6;
        } else {
            strcpy(materials[i]->materialName, "Nickel");
```

```
            materials[i]->permeability  = 6.5e-6;
            materials[i]->saturation  = 0.48e6;
        }
    }

    for (int i = 0; i < numMaterials;  i++) {
        printMaterialProperties(materials[i]);
    }

    for (int i = 0; i < numMaterials;  i++) {
        free(materials[i]);
    }
    free(materials);

    return 0;
}

//11.

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

typedef struct {
    double density;
    double temperature;
    double electricField;
} Plasma;

typedef union {
    double density;
    double temperature;
```

```
} PlasmaData;

void printPlasmaData(Plasma *plasma, const char *type) {
    printf("Plasma Type: %s\n", type);
    printf("Density: %.2f particles/m^3\n", plasma->density);
    printf("Temperature: %.2f K\n", plasma->temperature);
    printf("Electric Field: %.2f V/m\n", plasma->electricField);
}

int main() {
    int numPlasmaTypes = 2;
    Plasma **plasmas = (Plasma **)malloc(numPlasmaTypes * sizeof(Plasma *));
    if (plasmas == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numPlasmaTypes; i++) {
        plasmas[i] = (Plasma *)malloc(sizeof(Plasma));
        if (plasmas[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            plasmas[i]->density = 1.0e19;
            plasmas[i]->temperature = 15000;
            plasmas[i]->electricField = 2.5e3;
        } else {
            plasmas[i]->density = 5.0e18;
            plasmas[i]->temperature = 10000;
```

```
            plasmas[i]->electricField = 3.0e3;
        }
    }

    const char *types[] = {"Ionized Plasma", "Neutral Plasma"};
    for (int i = 0; i < numPlasmaTypes; i++) {
        printPlasmaData(plasmas[i], types[i]);
    }

    for (int i = 0; i < numPlasmaTypes; i++) {
        free(plasmas[i]);
    }
    free(plasmas);

    return 0;
}

//12.

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

typedef struct {
    double initialVelocity;
    double acceleration;
    double displacement;
} Kinematics;

typedef union {
    double velocity;
```

```
    double displacement;
} MotionData;

void printKinematicsData(Kinematics *kinematics, const char *description) {
    printf("Motion Description: %s\n", description);
    printf("Initial Velocity: %.2f m/s\n", kinematics->initialVelocity);
    printf("Acceleration: %.2f m/s^2\n", kinematics->acceleration);
    printf("Displacement: %.2f m\n", kinematics->displacement);
}

int main() {
    int numObjects = 2;
    Kinematics **objects = (Kinematics **)malloc(numObjects * sizeof(Kinematics *));
    if (objects == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numObjects; i++) {
        objects[i] = (Kinematics *)malloc(sizeof(Kinematics));
        if (objects[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            objects[i]->initialVelocity = 0.0;
            objects[i]->acceleration = 9.8;
            objects[i]->displacement = 100.0;
        } else {
            objects[i]->initialVelocity = 10.0;
```

```
            objects[i]->acceleration = 3.0;
            objects[i]->displacement  = 50.0;
        }
    }

    const char *descriptions[] = {"Free Fall", "Accelerating Object"};
    for (int i = 0; i < numObjects; i++) {
        printKinematicsData(objects[i], descriptions[i]);
    }

    for (int i = 0; i < numObjects; i++) {
        free(objects[i]);
    }
    free(objects);

    return 0;
}

//13.

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

typedef struct {
    char elementName[50];
    double wavelength;
    double intensity;
} SpectralLine;

typedef union {
```

```
    double wavelength;
    double intensity;
} SpectralData;

void printSpectralLine(SpectralLine *line) {
    printf("Element: %s\n", line->elementName);
    printf("Wavelength: %.2f nm\n", line->wavelength);
    printf("Intensity: %.2f units\n", line->intensity);
}

int main() {
    int numLines = 2;
    SpectralLine **lines = (SpectralLine **)malloc(numLines * sizeof(SpectralLine *));
    if (lines == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numLines; i++) {
        lines[i] = (SpectralLine *)malloc(sizeof(SpectralLine));
        if (lines[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            strcpy(lines[i]->elementName, "Hydrogen");
            lines[i]->wavelength = 656.3;
            lines[i]->intensity = 10.0;
        } else {
            strcpy(lines[i]->elementName, "Helium");
```

```
            lines[i]->wavelength = 587.6;
            lines[i]->intensity = 7.5;
        }
    }

    for (int i = 0; i < numLines; i++) {
        printSpectralLine(lines[i]);
    }

    for (int i = 0; i < numLines; i++) {
        free(lines[i]);
    }
    free(lines);

    return 0;
}

//14.

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

typedef struct {
    double mass;
    double velocity;
    double angle;
} Projectile;

typedef union {
    double velocity;
```

```
    double displacement;
} MotionParameter;

void printProjectileMotion(Projectile *projectile) {
    printf("Projectile Mass: %.2f kg\n", projectile->mass);
    printf("Initial Velocity: %.2f m/s\n", projectile->velocity);
    printf("Launch Angle: %.2f degrees\n", projectile->angle);
}

int main() {
    int numProjectiles = 2;
    Projectile **projectiles = (Projectile **)malloc(numProjectiles * sizeof(Projectile *));
    if (projectiles == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numProjectiles; i++) {
        projectiles[i] = (Projectile *)malloc(sizeof(Projectile));
        if (projectiles[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            projectiles[i]->mass = 1.0;
            projectiles[i]->velocity = 30.0;
            projectiles[i]->angle = 45.0;
        } else {
            projectiles[i]->mass = 0.5;
            projectiles[i]->velocity = 20.0;
```

```
            projectiles[i]->angle = 60.0;
        }
    }

    for (int i = 0; i < numProjectiles; i++) {
        printProjectileMotion(projectiles[i]);
    }

    for (int i = 0; i < numProjectiles; i++) {
        free(projectiles[i]);
    }
    free(projectiles);

    return 0;
}

//15.

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

typedef struct {
    double stress;
    double strain;
    double modulus;
} StressStrain;

typedef union {
    double stress;
    double modulus;
```

```
} MaterialProperties;

void printStressStrain(StressStrain *material) {
    printf("Stress: %.2f Pa\n", material->stress);
    printf("Strain: %.2f\n", material->strain);
    printf("Modulus: %.2f Pa\n", material->modulus);
}

int main() {
    int numMaterials = 2;
    StressStrain **materials = (StressStrain **)malloc(numMaterials * sizeof(StressStrain *));
    if (materials == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < numMaterials; i++) {
        materials[i] = (StressStrain *)malloc(sizeof(StressStrain));
        if (materials[i] == NULL) {
            printf("Memory allocation failed!\n");
            return 1;
        }

        if (i == 0) {
            materials[i]->stress = 250.0e6;
            materials[i]->strain = 0.02;
            materials[i]->modulus = 125.0e9;
        } else {
            materials[i]->stress = 150.0e6;
            materials[i]->strain = 0.015;
            materials[i]->modulus = 100.0e9;
```

```
        }
    }

    for (int i = 0; i < numMaterials;  i++) {
        printStressStrain(materials[i]);
    }

    for (int i = 0; i < numMaterials;  i++) {
        free(materials[i]);
    }
    free(materials);

    return 0;
}
```