--Solutions--

1.

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

typedef struct {
    double mass;
    double position[2];
    double velocity[2];
} Particle;

typedef union {
    double gravitational;
    double electric;
    double magnetic;
} Force;

int main() {
    int numParticles = 5;
    Particle* particles = (Particle*)malloc(numParticles * sizeof(Particle));

    for (int i = 0; i < numParticles; i++) {
        particles[i].mass = 1.0;
        particles[i].position[0] = i * 2.0;
        particles[i].position[1] = i * 3.0;
        particles[i].velocity[0] = 1.0;
        particles[i].velocity[1] = 1.0;
    }
    for (int i = 0; i < numParticles; i++) {
        printf("Particle %d: Position(%.2f, %.2f) Velocity(%.2f, %.2f)\n",
               i, particles[i].position[0], particles[i].position[1],
               particles[i].velocity[0], particles[i].velocity[1]);
    }

    free(particles);
    return 0;
}
```

2.

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

typedef struct {
    double electricField[3];
    double magneticField[3];
    double position[3];
} FieldPoint;

typedef union {
    double electric;
    double magnetic;
} FieldComponent;

int main() {
    int gridPoints = 5;
    FieldPoint* fields = (FieldPoint*)malloc(gridPoints * sizeof(FieldPoint));
    for (int i = 0; i < gridPoints; i++) {
        fields[i].electricField[0] = 1.0;
        fields[i].electricField[1] = 2.0;
        fields[i].electricField[2] = 3.0;
        fields[i].magneticField[0] = 0.5;
        fields[i].magneticField[1] = 1.5;
        fields[i].magneticField[2] = 2.5;
        fields[i].position[0] = i;
        fields[i].position[1] = i * 2.0;
        fields[i].position[2] = i * 3.0;
    }

    for (int i = 0; i < gridPoints; i++) {
        printf("Point %d: E(%.2f, %.2f, %.2f) B(%.2f, %.2f, %.2f) Position(%.2f, %.2f, %.2f)\n",
               i, fields[i].electricField[0], fields[i].electricField[1], fields[i].electricField[2],
               fields[i].magneticField[0], fields[i].magneticField[1], fields[i].magneticField[2],
               fields[i].position[0], fields[i].position[1], fields[i].position[2]);
    }

    free(fields);
    return 0;
}
```

3.

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

typedef struct {
    char elementName[20];
    double energyLevels[5];
    double transitionProbabilities[5];
} Atom;

int main() {
    int numAtoms = 3;
    Atom* atoms = (Atom*)malloc(numAtoms * sizeof(Atom));

    for (int i = 0; i < numAtoms; i++) {
        sprintf(atoms[i].elementName, "Element %d", i);
        for (int j = 0; j < 5; j++) {
            atoms[i].energyLevels[j] = 1.0 + j;
            atoms[i].transitionProbabilities[j] = 0.1 * (j + 1);
        }
    }

    for (int i = 0; i < numAtoms; i++) {
        printf("Atom %d: %s\n", i, atoms[i].elementName);
        for (int j = 0; j < 5; j++) {
            printf("  Energy Level %.2f Transition Probability %.2f\n", atoms[i].energyLevels[j], atoms[i].transitionProbabilities[j]);
        }
    }

    free(atoms);
    return 0;
}
```

4.

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

typedef struct {
    double wavefunctionAmplitude;
    double phase;
    double energy;
} QuantumState;

int main() {
    int numStates = 3;
    QuantumState* states = (QuantumState*)malloc(numStates * sizeof(QuantumState));

    for (int i = 0; i < numStates; i++) {
        states[i].wavefunctionAmplitude = 1.0;
        states[i].phase = 0.0;
        states[i].energy = 0.5 * i;
    }

    for (int i = 0; i < numStates; i++) {
        printf("State %d: Amplitude %.2f Phase %.2f Energy %.2f\n",
               i, states[i].wavefunctionAmplitude, states[i].phase, states[i].energy);
    }

    free(states);
    return 0;
}
```

5.

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

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

int main() {
    int numElements = 3;
    OpticalElement* elements = (OpticalElement*)malloc(numElements * sizeof(OpticalElement));

    for (int i = 0; i < numElements; i++) {
        elements[i].refractiveIndex = 1.5;
        elements[i].focalLength = 10.0;
    }

    for (int i = 0; i < numElements; i++) {
        printf("Element %d: Refractive Index %.2f Focal Length %.2f\n",
                i, elements[i].refractiveIndex, elements[i].focalLength);
    }

    free(elements);
    return 0;
}
```

6.

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

```
typedef struct {
    double mass;
    double position[2];
    double velocity[2];
} Particle;

typedef union {
    double gravitational;
    double electric;
    double magnetic;
} Force;

int main() {
    int numParticles = 5;
    Particle* particles = (Particle*)malloc(numParticles * sizeof(Particle));

    for (int i = 0; i < numParticles; i++) {
        particles[i].mass = 1.0;
        particles[i].position[0] = i * 2.0;
        particles[i].position[1] = i * 3.0;
        particles[i].velocity[0] = 1.0;
        particles[i].velocity[1] = 1.0;
    }

    for (int i = 0; i < numParticles; i++) {
        printf("Particle %d: Position(%.2f, %.2f) Velocity(%.2f, %.2f)\n",
               i, particles[i].position[0], particles[i].position[1],
               particles[i].velocity[0], particles[i].velocity[1]);
    }

    free(particles);
    return 0;
}
```

7.

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

typedef struct {
    double electricField[3];
    double magneticField[3];
    double position[3];
} FieldPoint;

typedef union {
    double electric;
    double magnetic;
} FieldComponent;

int main() {
    int gridPoints = 5;
    FieldPoint* fields = (FieldPoint*)malloc(gridPoints * sizeof(FieldPoint));

    for (int i = 0; i < gridPoints; i++) {
        fields[i].electricField[0] = 1.0;
        fields[i].electricField[1] = 2.0;
        fields[i].electricField[2] = 3.0;
        fields[i].magneticField[0] = 0.5;
        fields[i].magneticField[1] = 1.5;
        fields[i].magneticField[2] = 2.5;
        fields[i].position[0] = i;
        fields[i].position[1] = i * 2.0;
        fields[i].position[2] = i * 3.0;
    }

    for (int i = 0; i < gridPoints; i++) {
        printf("Point %d: E(%.2f, %.2f, %.2f) B(%.2f, %.2f, %.2f) Position(%.2f, %.2f, %.2f)\n",
               i, fields[i].electricField[0], fields[i].electricField[1], fields[i].electricField[2],
               fields[i].magneticField[0], fields[i].magneticField[1], fields[i].magneticField[2],
               fields[i].position[0], fields[i].position[1], fields[i].position[2]);
    }

    free(fields);
    return 0;
}
```

8.

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

typedef struct {
    char elementName[20];
    double energyLevels[5];
    double transitionProbabilities[5];
} Atom;
```

```
int main() {
    int numAtoms = 3;
    Atom* atoms = (Atom*)malloc(numAtoms * sizeof(Atom));

    for (int i = 0; i < numAtoms; i++) {
        sprintf(atoms[i].elementName, "Element %d", i);
        for (int j = 0; j < 5; j++) {
            atoms[i].energyLevels[j] = 1.0 + j;
            atoms[i].transitionProbabilities[j] = 0.1 * (j + 1);
        }
    }

    for (int i = 0; i < numAtoms; i++) {
        printf("Atom %d: %s\n", i, atoms[i].elementName);
        for (int j = 0; j < 5; j++) {
            printf("  Energy Level %.2f Transition Probability %.2f\n", atoms[i].energyLevels[j], atoms[i].transitionProbabilities[j]);
        }
    }

    free(atoms);
    return 0;
}
```

9.

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

typedef struct {
    double wavefunctionAmplitude;
    double phase;
    double energy;
} QuantumState;

int main() {
    int numStates = 3;
    QuantumState* states = (QuantumState*)malloc(numStates * sizeof(QuantumState));

    for (int i = 0; i < numStates; i++) {
        states[i].wavefunctionAmplitude = 1.0;
        states[i].phase = 0.0;
        states[i].energy = 0.5 * i;
    }

    for (int i = 0; i < numStates; i++) {
        printf("State %d: Amplitude %.2f Phase %.2f Energy %.2f\n",
               i, states[i].wavefunctionAmplitude, states[i].phase, states[i].energy);
    }

    free(states);
    return 0;
}
```

10.

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

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

int main() {
    int numElements = 3;
    OpticalElement* elements = (OpticalElement*)malloc(numElements * sizeof(OpticalElement));

    for (int i = 0; i < numElements; i++) {
        elements[i].refractiveIndex = 1.5;
        elements[i].focalLength = 10.0;
    }

    for (int i = 0; i < numElements; i++) {
        printf("Element %d: Refractive Index %.2f Focal Length %.2f\n",
                i, elements[i].refractiveIndex, elements[i].focalLength);
    }

    free(elements);
    return 0;
}
```

11.

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

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

int main() {
    int numStates = 3;
    ThermodynamicState* states = (ThermodynamicState*)malloc(numStates *
sizeof(ThermodynamicState));

    for (int i = 0; i < numStates; i++) {
        states[i].pressure = 101325 + i * 1000;
        states[i].volume = 1.0 + i * 0.5;
        states[i].temperature = 300 + i * 10;
        states[i].entropy = 1.5 + i * 0.2;
    }

    for (int i = 0; i < numStates; i++) {
        printf("State %d: P %.2f V %.2f T %.2f S %.2f\n",
               i, states[i].pressure, states[i].volume, states[i].temperature, states[i].entropy);
    }

    free(states);
    return 0;
}
```

12.

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

```
typedef struct {
    char reactant[20];
    char product[20];
    double energyReleased;
} NuclearReaction;

int main() {
    int numReactions = 3;
    NuclearReaction* reactions = (NuclearReaction*)malloc(numReactions * sizeof(NuclearReaction));

    for (int i = 0; i < numReactions; i++) {
        sprintf(reactions[i].reactant, "Reactant %d", i);
        sprintf(reactions[i].product, "Product %d", i);
        reactions[i].energyReleased = 5.0 + i * 2.0;
    }

    for (int i = 0; i < numReactions; i++) {
        printf("Reaction %d: %s -> %s Energy Released %.2f\n",
               i, reactions[i].reactant, reactions[i].product, reactions[i].energyReleased);
    }

    free(reactions);
    return 0;
}
```

13.

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

```
typedef struct {
    double mass;
    double position[3];
    double fieldStrength;
} GravitationalObject;

int main() {
    int numObjects = 3;
    GravitationalObject* objects = (GravitationalObject*)malloc(numObjects *
sizeof(GravitationalObject));

    for (int i = 0; i < numObjects; i++) {
        objects[i].mass = 5.0 + i * 2.0;
        objects[i].position[0] = i;
        objects[i].position[1] = i * 2.0;
        objects[i].position[2] = i * 3.0;
        objects[i].fieldStrength = 9.8 * objects[i].mass;
    }

    for (int i = 0; i < numObjects; i++) {
        printf("Object %d: Mass %.2f Position(%.2f, %.2f, %.2f) Field Strength %.2f\n",
               i, objects[i].mass, objects[i].position[0], objects[i].position[1],
               objects[i].position[2], objects[i].fieldStrength);
    }

    free(objects);
    return 0;
}
```

14.

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

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

int main() {
    int numWaves = 3;
    Wave* waves = (Wave*)malloc(numWaves * sizeof(Wave));

    for (int i = 0; i < numWaves; i++) {
        waves[i].amplitude = 1.0 + i * 0.5;
        waves[i].wavelength = 500.0 + i * 50.0;
        waves[i].phase = 0.0 + i * 0.1;
    }

    for (int i = 0; i < numWaves; i++) {
        printf("Wave %d: Amplitude %.2f Wavelength %.2f Phase %.2f\n",
               i, waves[i].amplitude, waves[i].wavelength, waves[i].phase);
    }

    free(waves);
    return 0;
}
```

15.

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

typedef struct {
    char materialName[20];
    double permeability;
    double saturation;
} MagneticMaterial;

int main() {
    int numMaterials = 3;
    MagneticMaterial* materials = (MagneticMaterial*)malloc(numMaterials * sizeof(MagneticMaterial));

    for (int i = 0; i < numMaterials; i++) {
        sprintf(materials[i].materialName, "Material %d", i);
        materials[i].permeability = 4.0 + i * 0.2;
        materials[i].saturation = 1.2 + i * 0.5;
    }

    for (int i = 0; i < numMaterials; i++) {
        printf("Material %d: %s Permeability %.2f Saturation %.2f\n",
               i, materials[i].materialName, materials[i].permeability, materials[i].saturation);
    }

    free(materials);
    return 0;
}
```