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

#define MAX_ALLOYS 100

typedef enum {
    METAL_A,
    METAL_B,
    METAL_C,
    METAL_D,
    NUM_METALS
} MetalType;

typedef union {
    float percentages[NUM_METALS];
} Composition;

typedef struct {
    int sampleID;
    char name[50];
    Composition comp;
} AlloySample;

void inputAlloySample(AlloySample *sample) {
    printf("Enter sample ID: ");
    scanf("%d", &sample->sampleID);
    printf("Enter alloy name: ");
    getchar();
    fgets(sample->name, sizeof(sample->name), stdin);
```

```
    sample->name[strcspn(sample->name, "\n")] = 0;

    for (int i = 0; i < NUM_METALS; i++) {
        printf("Enter percentage of Metal %d: ", i + 1);
        scanf("%f", &sample->comp.percentages[i]);
    }
}

void printAlloySample(const AlloySample *sample) {
    printf("\nAlloy Sample ID: %d\n", sample->sampleID);
    printf("Alloy Name: %s\n", sample->name);
    printf("Composition Details:\n");
    for (int i = 0; i < NUM_METALS; i++) {
        printf("  Metal %d: %.2f%%\n", i + 1, sample->comp.percentages[i]);
    }
}

void allocateAlloySamples(AlloySample ***samples, int *numAlloys) {
    printf("Enter the number of alloy samples: ");
    scanf("%d", numAlloys);

    *samples = (AlloySample **)malloc(*numAlloys * sizeof(AlloySample *));
    for (int i = 0; i < *numAlloys; i++) {
        (*samples)[i] = (AlloySample *)malloc(sizeof(AlloySample));
    }
}

void freeAlloySamples(AlloySample ***samples, int numAlloys) {
    for (int i = 0; i < numAlloys; i++) {
        free((*samples)[i]);
    }
```

```
    free(*samples);
}

int main() {
    AlloySample **alloySamples = NULL;
    int numAlloys = 0;

    allocateAlloySamples(&alloySamples, &numAlloys);

    for (int i = 0; i < numAlloys; i++) {
        printf("\nEnter details for Alloy Sample %d:\n", i + 1);
        inputAlloySample(alloySamples[i]);
    }

    for (int i = 0; i < numAlloys; i++) {
        printAlloySample(alloySamples[i]);
    }

    freeAlloySamples(&alloySamples, numAlloys);

    return 0;
}

//2.

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

#define MAX_PROCESSES 100
```

```
typedef struct {
    int processID;
    float temperature;
    float duration;
    float coolingRate;
} HeatTreatmentProcess;

void inputHeatTreatmentProcess(HeatTreatmentProcess *process) {
    printf("Enter Process ID: ");
    scanf("%d", &process->processID);
    printf("Enter Process Temperature (Celsius): ");
    scanf("%f", &process->temperature);
    printf("Enter Process Duration (minutes): ");
    scanf("%f", &process->duration);
    printf("Enter Cooling Rate (degrees per minute): ");
    scanf("%f", &process->coolingRate);
}

void printHeatTreatmentProcess(const HeatTreatmentProcess *process) {
    printf("\nHeat Treatment Process ID: %d\n", process->processID);
    printf("Temperature: %.2f°C\n", process->temperature);
    printf("Duration: %.2f minutes\n", process->duration);
    printf("Cooling Rate: %.2f degrees per minute\n", process->coolingRate);
}

void allocateHeatTreatmentProcesses(HeatTreatmentProcess ***processes, int *numProcesses) {
    printf("Enter the number of heat treatment processes: ");
    scanf("%d", numProcesses);
    *processes = (HeatTreatmentProcess **)malloc(*numProcesses * sizeof(HeatTreatmentProcess
*));
    for (int i = 0; i < *numProcesses; i++) {
```

```
        (*processes)[i] = (HeatTreatmentProcess *)malloc(sizeof(HeatTreatmentProcess));
    }
}

void freeHeatTreatmentProcesses(HeatTreatmentProcess ***processes, int numProcesses) {
    for (int i = 0; i < numProcesses; i++) {
        free((*processes)[i]);
    }
    free(*processes);
}

int main() {
    HeatTreatmentProcess **heatTreatmentProcesses = NULL;
    int numProcesses = 0;

    allocateHeatTreatmentProcesses(&heatTreatmentProcesses, &numProcesses);

    for (int i = 0; i < numProcesses; i++) {
        printf("\nEnter details for Heat Treatment Process %d:\n", i + 1);
        inputHeatTreatmentProcess(heatTreatmentProcesses[i]);
    }

    for (int i = 0; i < numProcesses; i++) {
        printHeatTreatmentProcess(heatTreatmentProcesses[i]);
    }

    freeHeatTreatmentProcesses(&heatTreatmentProcesses, numProcesses);

    return 0;
}
```

```
//3.

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

#define MAX_TESTS 100

typedef enum {
    TENSILE_STRENGTH,
    HARDNESS,
    ELONGATION
} QualityMetricType;

typedef union {
    float tensileStrength;
    float hardness;
    float elongation;
} QualityMetric;

typedef struct {
    int testID;
    char testType[50];
    QualityMetric result;
} TestRecord;

void inputTestRecord(TestRecord *record) {
    printf("Enter Test ID: ");
    scanf("%d", &record->testID);
    printf("Enter Test Type (e.g., Tensile Strength, Hardness, Elongation): ");
    getchar();
```

```
    fgets(record->testType, sizeof(record->testType), stdin);
    record->testType[strcspn(record->testType, "\n")] = 0;
    printf("Enter result value: ");
    if (strstr(record->testType, "Tensile Strength")) {
        scanf("%f", &record->result.tensileStrength);
    } else if (strstr(record->testType, "Hardness")) {
        scanf("%f", &record->result.hardness);
    } else if (strstr(record->testType, "Elongation")) {
        scanf("%f", &record->result.elongation);
    }
}

void printTestRecord(const TestRecord *record) {
    printf("\nTest ID: %d\n", record->testID);
    printf("Test Type: %s\n", record->testType);
    if (strstr(record->testType, "Tensile Strength")) {
        printf("Tensile Strength: %.2f\n", record->result.tensileStrength);
    } else if (strstr(record->testType, "Hardness")) {
        printf("Hardness: %.2f\n", record->result.hardness);
    } else if (strstr(record->testType, "Elongation")) {
        printf("Elongation: %.2f\n", record->result.elongation);
    }
}

void allocateTestRecords(TestRecord ***records, int *numTests) {
    printf("Enter the number of test records: ");
    scanf("%d", numTests);
    *records = (TestRecord **)malloc(*numTests * sizeof(TestRecord *));
    for (int i = 0; i < *numTests; i++) {
        (*records)[i] = (TestRecord *)malloc(sizeof(TestRecord));
    }
```

```
}

void freeTestRecords(TestRecord ***records, int numTests) {
    for (int i = 0; i < numTests; i++) {
        free((*records)[i]);
    }
    free(*records);
}

int main() {
    TestRecord **testRecords = NULL;
    int numTests = 0;

    allocateTestRecords(&testRecords, &numTests);

    for (int i = 0; i < numTests; i++) {
        printf("\nEnter details for Test Record %d:\n", i + 1);
        inputTestRecord(testRecords[i]);
    }

    for (int i = 0; i < numTests; i++) {
        printTestRecord(testRecords[i]);
    }

    freeTestRecords(&testRecords, numTests);

    return 0;
}

//4.
```

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

#define MAX_MATERIALS 100
#define MAX_CYCLES 10

typedef struct {
    int materialID;
    char name[50];
    float enduranceLimit;
} Material;

typedef struct {
    int cycleNumber;
    float stress;
} StressCycleData;

void inputMaterialDetails(Material *material) {
    printf("Enter Material ID: ");
    scanf("%d", &material->materialID);
    printf("Enter Material Name: ");
    getchar();
    fgets(material->name, sizeof(material->name), stdin);
    material->name[strcspn(material->name, "\n")] = 0;
    printf("Enter Endurance Limit: ");
    scanf("%f", &material->enduranceLimit);
}

void inputStressCycleData(StressCycleData *data) {
    printf("Enter Cycle Number: ");
```

```
    scanf("%d", &data->cycleNumber);
    printf("Enter Stress for Cycle %d: ", data->cycleNumber);
    scanf("%f", &data->stress);
}

void printMaterialDetails(const Material *material) {
    printf("\nMaterial ID: %d\n", material->materialID);
    printf("Material Name: %s\n", material->name);
    printf("Endurance Limit: %.2f\n", material->enduranceLimit);
}

void printStressCycleData(const StressCycleData *data) {
    printf("Cycle %d: Stress = %.2f\n", data->cycleNumber, data->stress);
}

void allocateMaterialData(Material ***materials, int *numMaterials, StressCycleData ****cycleData,
int *numCycles) {
    printf("Enter the number of materials: ");
    scanf("%d", numMaterials);
    *materials = (Material **)malloc(*numMaterials * sizeof(Material *));
    *cycleData = (StressCycleData ***)malloc(*numMaterials * sizeof(StressCycleData **));

    for (int i = 0; i < *numMaterials; i++) {
        (*materials)[i] = (Material *)malloc(sizeof(Material));
        printf("Enter the number of cycles for Material %d: ", i + 1);
        scanf("%d", numCycles);
        (*cycleData)[i] = (StressCycleData **)malloc(*numCycles * sizeof(StressCycleData *));
        for (int j = 0; j < *numCycles; j++) {
            (*cycleData)[i][j] = (StressCycleData *)malloc(sizeof(StressCycleData));
        }
    }
```

```
}

void freeMaterialData(Material ***materials, int numMaterials, StressCycleData ****cycleData, int *numCycles) {
    for (int i = 0; i < numMaterials; i++) {
        free((*materials)[i]);
        for (int j = 0; j < *numCycles; j++) {
            free((*cycleData)[i][j]);
        }
        free((*cycleData)[i]);
    }
    free(*materials);
    free(*cycleData);
}

int main() {
    Material **materials = NULL;
    StressCycleData ***cycleData = NULL;
    int numMaterials = 0, numCycles = 0;

    allocateMaterialData(&materials, &numMaterials, &cycleData, &numCycles);

    for (int i = 0; i < numMaterials; i++) {
        printf("\nEnter details for Material %d:\n", i + 1);
        inputMaterialDetails(materials[i]);
        for (int j = 0; j < numCycles; j++) {
            printf("\nEnter Stress Cycle Data for Material %d, Cycle %d:\n", i + 1, j + 1);
            inputStressCycleData(cycleData[i][j]);
        }
    }
```

```
    for (int i = 0; i < numMaterials;  i++) {
        printMaterialDetails(materials[i]);
        for (int j = 0; j < numCycles; j++) {
            printStressCycleData(cycleData[i][j]);
        }
    }

    freeMaterialData(&materials,  numMaterials,  &cycleData, &numCycles);

    return 0;
}

//5.

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

#define MAX_CASTINGS 100
#define MAX_EQUIPMENT 10

typedef enum {
    DIMENSIONS,
    THERMAL_CONDUCTIVITY
} MoldPropertyType;

typedef union {
    float dimensions[3];   // Length, width, height
    float thermalConductivity;
} MoldProperties;
```

```
typedef struct {
    int castingID;
    float weight;
    char material[50];
} Casting;

typedef struct {
    int equipmentID;
    char equipmentName[50];
    float capacity;
} Equipment;

void inputCastingDetails(Casting *casting) {
    printf("Enter Casting ID: ");
    scanf("%d", &casting->castingID);
    printf("Enter Casting Weight (kg): ");
    scanf("%f", &casting->weight);
    printf("Enter Material: ");
    getchar();
    fgets(casting->material, sizeof(casting->material), stdin);
    casting->material[strcspn(casting->material, "\n")] = 0;
}

void inputMoldProperties(MoldProperties *properties) {
    int choice;
    printf("Select Mold Property Type:\n");
    printf("1. Dimensions (Length, Width, Height)\n");
    printf("2. Thermal Conductivity\n");
    scanf("%d", &choice);

    if (choice == 1) {
```

```
        printf("Enter Dimensions (Length, Width, Height): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &properties->dimensions[i]);
        }
    } else if (choice == 2) {
        printf("Enter Thermal Conductivity: ");
        scanf("%f", &properties->thermalConductivity);
    }
}

void inputEquipmentData(Equipment *equipment) {
    printf("Enter Equipment ID: ");
    scanf("%d", &equipment->equipmentID);
    printf("Enter Equipment Name: ");
    getchar();
    fgets(equipment->equipmentName, sizeof(equipment->equipmentName), stdin);
    equipment->equipmentName[strcspn(equipment->equipmentName, "\n")] = 0;
    printf("Enter Equipment Capacity (tons): ");
    scanf("%f", &equipment->capacity);
}

void printCastingDetails(const Casting *casting) {
    printf("\nCasting ID: %d\n", casting->castingID);
    printf("Weight: %.2f kg\n", casting->weight);
    printf("Material: %s\n", casting->material);
}

void printMoldProperties(const MoldProperties *properties) {
    printf("Mold Properties:\n");
    if (properties->dimensions[0] != 0 && properties->dimensions[1] != 0 && properties-
>dimensions[2] != 0) {
```

```
        printf("Dimensions (Length, Width, Height): %.2f, %.2f, %.2f\n",
            properties->dimensions[0], properties->dimensions[1], properties->dimensions[2]);
    } else {
        printf("Thermal Conductivity: %.2f\n", properties->thermalConductivity);
    }
}

void printEquipmentData(const Equipment *equipment) {
    printf("\nEquipment ID: %d\n", equipment->equipmentID);
    printf("Equipment Name: %s\n", equipment->equipmentName);
    printf("Capacity: %.2f tons\n", equipment->capacity);
}

void allocateCastingRecords(Casting ***castings, int *numCastings) {
    printf("Enter the number of castings: ");
    scanf("%d", numCastings);
    *castings = (Casting **)malloc(*numCastings * sizeof(Casting *));
    for (int i = 0; i < *numCastings; i++) {
        (*castings)[i] = (Casting *)malloc(sizeof(Casting));
    }
}

void allocateEquipmentData(Equipment **equipment, int *numEquipment) {
    printf("Enter the number of equipment: ");
    scanf("%d", numEquipment);
    *equipment = (Equipment *)malloc(*numEquipment * sizeof(Equipment));
}

void freeCastingRecords(Casting ***castings, int numCastings) {
    for (int i = 0; i < numCastings; i++) {
        free((*castings)[i]);
```

```
    }
    free(*castings);
}

void freeEquipmentData(Equipment **equipment) {
    free(*equipment);
}

int main() {
    Casting **castings = NULL;
    Equipment *equipment = NULL;
    int numCastings = 0, numEquipment = 0;

    allocateCastingRecords(&castings, &numCastings);
    allocateEquipmentData(&equipment, &numEquipment);

    for (int i = 0; i < numCastings; i++) {
        printf("\nEnter details for Casting %d:\n", i + 1);
        inputCastingDetails(castings[i]);
        MoldProperties properties = {0};
        inputMoldProperties(&properties);
        printCastingDetails(castings[i]);
        printMoldProperties(&properties);
    }

    for (int i = 0; i < numEquipment; i++) {
        printf("\nEnter details for Equipment %d:\n", i + 1);
        inputEquipmentData(&equipment[i]);
        printEquipmentData(&equipment[i]);
    }
```

```
    freeCastingRecords(&castings, numCastings);
    freeEquipmentData(&equipment);

    return 0;
}

//6.

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

#define MAX_SAMPLES 100
#define MAX_IMPURITIES 5

typedef enum {
    TRACE_ELEMENTS,
    OXIDES
} ImpurityType;

typedef union {
    float traceElements[MAX_IMPURITIES];
    float oxides[MAX_IMPURITIES];
} ImpurityData;

typedef struct {
    int sampleID;
    char type[50];
    float purity;
} MetalSample;
```

```
typedef struct {
    ImpurityType impurityType;
    ImpurityData impurityData;
} ImpurityRecord;

void inputSampleData(MetalSample *sample) {
    printf("Enter Sample ID: ");
    scanf("%d", &sample->sampleID);
    printf("Enter Sample Type: ");
    getchar();
    fgets(sample->type, sizeof(sample->type), stdin);
    sample->type[strcspn(sample->type, "\n")] = 0;
    printf("Enter Purity Percentage: ");
    scanf("%f", &sample->purity);
}

void inputImpurityData(ImpurityRecord *impurityRecord) {
    int choice;
    printf("Select Impurity Type:\n");
    printf("1. Trace Elements\n");
    printf("2. Oxides\n");
    scanf("%d", &choice);

    if (choice == 1) {
        impurityRecord->impurityType = TRACE_ELEMENTS;
        printf("Enter percentages for Trace Elements:\n");
        for (int i = 0; i < MAX_IMPURITIES; i++) {
            printf("Element %d: ", i + 1);
            scanf("%f", &impurityRecord->impurityData.traceElements[i]);
        }
    } else if (choice == 2) {
```

```
        impurityRecord->impurityType = OXIDES;
        printf("Enter percentages for Oxides:\n");
        for (int i = 0; i < MAX_IMPURITIES; i++) {
            printf("Oxide %d: ", i + 1);
            scanf("%f", &impurityRecord->impurityData.oxides[i]);
        }
    }
}

void printSampleData(const MetalSample *sample) {
    printf("\nSample ID: %d\n", sample->sampleID);
    printf("Sample Type: %s\n", sample->type);
    printf("Purity: %.2f%%\n", sample->purity);
}

void printImpurityData(const ImpurityRecord *impurityRecord) {
    printf("Impurity Type: ");
    if (impurityRecord->impurityType == TRACE_ELEMENTS) {
        printf("Trace Elements\n");
        for (int i = 0; i < MAX_IMPURITIES; i++) {
            printf("Trace Element %d: %.2f%%\n", i + 1, impurityRecord->impurityData.traceElements[i]);
        }
    } else if (impurityRecord->impurityType == OXIDES) {
        printf("Oxides\n");
        for (int i = 0; i < MAX_IMPURITIES; i++) {
            printf("Oxide %d: %.2f%%\n", i + 1, impurityRecord->impurityData.oxides[i]);
        }
    }
}

void allocateSampleRecords(MetalSample ***samples, int *numSamples) {
```

```
    printf("Enter the number of samples: ");
    scanf("%d", numSamples);
    *samples = (MetalSample **)malloc(*numSamples * sizeof(MetalSample *));
    for (int i = 0; i < *numSamples; i++) {
        (*samples)[i] = (MetalSample *)malloc(sizeof(MetalSample));
    }
}

void freeSampleRecords(MetalSample ***samples, int numSamples) {
    for (int i = 0; i < numSamples; i++) {
        free((*samples)[i]);
    }
    free(*samples);
}

int main() {
    MetalSample **samples = NULL;
    ImpurityRecord impurityRecord;
    int numSamples = 0;

    allocateSampleRecords(&samples, &numSamples);

    for (int i = 0; i < numSamples; i++) {
        printf("\nEnter details for Sample %d:\n", i + 1);
        inputSampleData(samples[i]);
        inputImpurityData(&impurityRecord);
        printSampleData(samples[i]);
        printImpurityData(&impurityRecord);
    }

    freeSampleRecords(&samples, numSamples);
```

```
    return 0;
}

//7.

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

#define MAX_TESTS 100
#define MAX_RESULTS 10

typedef struct {
    int testID;
    int duration;
    char environment[50];
} CorrosionTest;

typedef struct {
    float testResults[MAX_RESULTS];
    char conditions[100];
} TestResults;

void inputTestDetails(CorrosionTest  *test) {
    printf("Enter Test ID: ");
    scanf("%d", &test->testID);
    printf("Enter Test Duration (days): ");
    scanf("%d", &test->duration);
    printf("Enter Environment Conditions: ");
    getchar();
```

```
    fgets(test->environment, sizeof(test->environment), stdin);
    test->environment[strcspn(test->environment, "\n")] = 0;
}

void inputTestResults(TestResults *results) {
    printf("Enter Test Conditions: ");
    fgets(results->conditions, sizeof(results->conditions), stdin);
    results->conditions[strcspn(results->conditions, "\n")] = 0;
    for (int i = 0; i < MAX_RESULTS; i++) {
        printf("Enter result for condition %d: ", i + 1);
        scanf("%f", &results->testResults[i]);
    }
}

void printTestDetails(const CorrosionTest *test) {
    printf("\nTest ID: %d\n", test->testID);
    printf("Test Duration: %d days\n", test->duration);
    printf("Environment: %s\n", test->environment);
}

void printTestResults(const TestResults *results) {
    printf("Test Conditions: %s\n", results->conditions);
    for (int i = 0; i < MAX_RESULTS; i++) {
        printf("Result %d: %.2f\n", i + 1, results->testResults[i]);
    }
}

void allocateTestRecords(CorrosionTest ***tests, int *numTests, TestResults ***results) {
    printf("Enter the number of corrosion tests: ");
    scanf("%d", numTests);
    *tests = (CorrosionTest **)malloc(*numTests * sizeof(CorrosionTest *));
```

```
    *results = (TestResults **)malloc(*numTests * sizeof(TestResults *));
    for (int i = 0; i < *numTests; i++) {
        (*tests)[i] = (CorrosionTest *)malloc(sizeof(CorrosionTest));
        (*results)[i] = (TestResults *)malloc(sizeof(TestResults));
    }
}

void freeTestRecords(CorrosionTest ***tests, int *numTests, TestResults ***results) {
    for (int i = 0; i < *numTests; i++) {
        free((*tests)[i]);
        free((*results)[i]);
    }
    free(*tests);
    free(*results);
}

int main() {
    CorrosionTest **tests = NULL;
    TestResults **results = NULL;
    int numTests = 0;

    allocateTestRecords(&tests, &numTests, &results);

    for (int i = 0; i < numTests; i++) {
        printf("\nEnter details for Test %d:\n", i + 1);
        inputTestDetails(tests[i]);
        inputTestResults(results[i]);
        printTestDetails(tests[i]);
        printTestResults(results[i]);
    }
```

```
    freeTestRecords(&tests, &numTests, &results);

    return 0;
}

//8.

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

#define MAX_PARAMETERS 100
#define MAX_OUTCOMES 10

typedef enum {
    MIG,
    TIG,
    ARC
} WeldingType;

typedef union {
    float migData[3];  // MIG: Voltage, Current, Speed
    float tigData[3];  // TIG: Voltage, Current, Speed
    float arcData[3];  // Arc: Voltage, Current, Speed
} WeldingData;

typedef struct {
    int paramID;
    float voltage;
    float current;
    float speed;
```

```
} WeldingParameterSet;

typedef struct {
    WeldingType weldingType;
    WeldingData weldingData;
} WeldingTestOutcome;

void inputWeldingParameterSet(WeldingParameterSet *paramSet) {
    printf("Enter Parameter ID: ");
    scanf("%d", &paramSet->paramID);
    printf("Enter Voltage (V): ");
    scanf("%f", &paramSet->voltage);
    printf("Enter Current (A): ");
    scanf("%f", &paramSet->current);
    printf("Enter Speed (mm/min): ");
    scanf("%f", &paramSet->speed);
}

void inputWeldingTestOutcome(WeldingTestOutcome *outcome) {
    int choice;
    printf("Select Welding Type:\n");
    printf("1. MIG\n");
    printf("2. TIG\n");
    printf("3. Arc\n");
    scanf("%d", &choice);

    if (choice == 1) {
        outcome->weldingType = MIG;
        printf("Enter MIG parameters (Voltage, Current, Speed): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &outcome->weldingData.migData[i]);
```

```
        }
    } else if (choice == 2) {
        outcome->weldingType = TIG;
        printf("Enter TIG parameters (Voltage, Current, Speed): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &outcome->weldingData.tigData[i]);
        }
    } else if (choice == 3) {
        outcome->weldingType = ARC;
        printf("Enter Arc parameters (Voltage, Current, Speed): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &outcome->weldingData.arcData[i]);
        }
    }
}

void printWeldingParameterSet(const WeldingParameterSet *paramSet) {
    printf("\nParameter ID: %d\n", paramSet->paramID);
    printf("Voltage: %.2f V\n", paramSet->voltage);
    printf("Current: %.2f A\n", paramSet->current);
    printf("Speed: %.2f mm/min\n", paramSet->speed);
}

void printWeldingTestOutcome(const WeldingTestOutcome *outcome) {
    printf("Welding Type: ");
    if (outcome->weldingType == MIG) {
        printf("MIG\n");
        printf("Voltage: %.2f V, Current: %.2f A, Speed: %.2f mm/min\n",
               outcome->weldingData.migData[0],
               outcome->weldingData.migData[1],
               outcome->weldingData.migData[2]);
```

```
    } else if (outcome->weldingType == TIG) {
        printf("TIG\n");
        printf("Voltage: %.2f V, Current: %.2f A, Speed: %.2f mm/min\n",
               outcome->weldingData.tigData[0],
               outcome->weldingData.tigData[1],
               outcome->weldingData.tigData[2]);
    } else if (outcome->weldingType == ARC) {
        printf("Arc\n");
        printf("Voltage: %.2f V, Current: %.2f A, Speed: %.2f mm/min\n",
               outcome->weldingData.arcData[0],
               outcome->weldingData.arcData[1],
               outcome->weldingData.arcData[2]);
    }
}

void allocateWeldingParameterSets(WeldingParameterSet ***paramSets, int *numParamSets) {
    printf("Enter the number of welding parameter sets: ");
    scanf("%d", numParamSets);
    *paramSets = (WeldingParameterSet **)malloc(*numParamSets * sizeof(WeldingParameterSet *));
    for (int i = 0; i < *numParamSets; i++) {
        (*paramSets)[i] = (WeldingParameterSet *)malloc(sizeof(WeldingParameterSet));
    }
}

void allocateWeldingTestOutcomes(WeldingTestOutcome ***testOutcomes, int *numTestOutcomes)
{
    printf("Enter the number of welding test outcomes: ");
    scanf("%d", numTestOutcomes);
    *testOutcomes = (WeldingTestOutcome **)malloc(*numTestOutcomes *
sizeof(WeldingTestOutcome *));
    for (int i = 0; i < *numTestOutcomes; i++) {
        (*testOutcomes)[i] = (WeldingTestOutcome *)malloc(sizeof(WeldingTestOutcome));
```

```
    }
}

void freeWeldingParameterSets(WeldingParameterSet ***paramSets, int numParamSets) {
    for (int i = 0; i < numParamSets; i++) {
        free((*paramSets)[i]);
    }
    free(*paramSets);
}

void freeWeldingTestOutcomes(WeldingTestOutcome ***testOutcomes, int numTestOutcomes) {
    for (int i = 0; i < numTestOutcomes; i++) {
        free((*testOutcomes)[i]);
    }
    free(*testOutcomes);
}

int main() {
    WeldingParameterSet **paramSets = NULL;
    WeldingTestOutcome **testOutcomes = NULL;
    int numParamSets = 0, numTestOutcomes = 0;

    allocateWeldingParameterSets(&paramSets, &numParamSets);
    allocateWeldingTestOutcomes(&testOutcomes, &numTestOutcomes);

    for (int i = 0; i < numParamSets; i++) {
        printf("\nEnter details for Parameter Set %d:\n", i + 1);
        inputWeldingParameterSet(paramSets[i]);
    }

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

```
        printf("\nEnter welding test outcome %d:\n", i + 1);
        inputWeldingTestOutcome(testOutcomes[i]);
    }

    for (int i = 0; i < numParamSets; i++) {
        printWeldingParameterSet(paramSets[i]);
    }

    for (int i = 0; i < numTestOutcomes; i++) {
        printWeldingTestOutcome(testOutcomes[i]);
    }

    freeWeldingParameterSets(&paramSets, numParamSets);
    freeWeldingTestOutcomes(&testOutcomes, numTestOutcomes);

    return 0;
}

//9.

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

#define MAX_SURFACES 100
#define MAX_MEASUREMENTS 10

typedef struct {
    int configID;
    char material[50];
    char measurementUnits[20];
```

```
} SurfaceFinishConfig;

typedef struct {
    float measurements[MAX_MEASUREMENTS];
    char surfaceType[50];
} SurfaceFinishMeasurements;

void inputSurfaceFinishConfig(SurfaceFinishConfig *config) {
    printf("Enter Configuration ID: ");
    scanf("%d", &config->configID);
    printf("Enter Material: ");
    getchar();
    fgets(config->material, sizeof(config->material), stdin);
    config->material[strcspn(config->material, "\n")] = 0;
    printf("Enter Measurement Units: ");
    fgets(config->measurementUnits, sizeof(config->measurementUnits), stdin);
    config->measurementUnits[strcspn(config->measurementUnits, "\n")] = 0;
}

void inputSurfaceFinishMeasurements(SurfaceFinishMeasurements *measurements) {
    printf("Enter Surface Type: ");
    getchar();
    fgets(measurements->surfaceType, sizeof(measurements->surfaceType), stdin);
    measurements->surfaceType[strcspn(measurements->surfaceType, "\n")] = 0;
    for (int i = 0; i < MAX_MEASUREMENTS; i++) {
        printf("Enter measurement %d: ", i + 1);
        scanf("%f", &measurements->measurements[i]);
    }
}

void printSurfaceFinishConfig(const SurfaceFinishConfig *config) {
```

```
    printf("\nConfiguration ID: %d\n", config->configID);
    printf("Material: %s\n", config->material);
    printf("Measurement Units: %s\n", config->measurementUnits);
}

void printSurfaceFinishMeasurements(const SurfaceFinishMeasurements *measurements) {
    printf("Surface Type: %s\n", measurements->surfaceType);
    for (int i = 0; i < MAX_MEASUREMENTS; i++) {
        printf("Measurement %d: %.2f\n", i + 1, measurements->measurements[i]);
    }
}

void allocateSurfaceFinishConfigs(SurfaceFinishConfig ***configs, int *numConfigs) {
    printf("Enter the number of surface finish configurations: ");
    scanf("%d", numConfigs);
    *configs = (SurfaceFinishConfig **)malloc(*numConfigs * sizeof(SurfaceFinishConfig *));
    for (int i = 0; i < *numConfigs; i++) {
        (*configs)[i] = (SurfaceFinishConfig *)malloc(sizeof(SurfaceFinishConfig));
    }
}

void allocateSurfaceFinishMeasurements(SurfaceFinishMeasurements ***measurements, int
*numMeasurements) {
    printf("Enter the number of surface finish measurements: ");
    scanf("%d", numMeasurements);
    *measurements = (SurfaceFinishMeasurements **)malloc(*numMeasurements *
sizeof(SurfaceFinishMeasurements *));
    for (int i = 0; i < *numMeasurements; i++) {
        (*measurements)[i] = (SurfaceFinishMeasurements
*)malloc(sizeof(SurfaceFinishMeasurements));
    }
}
```

```
void freeSurfaceFinishConfigs(SurfaceFinishConfig ***configs, int numConfigs) {
    for (int i = 0; i < numConfigs; i++) {
        free((*configs)[i]);
    }
    free(*configs);
}

void freeSurfaceFinishMeasurements(SurfaceFinishMeasurements ***measurements, int
numMeasurements) {
    for (int i = 0; i < numMeasurements; i++) {
        free((*measurements)[i]);
    }
    free(*measurements);
}

int main() {
    SurfaceFinishConfig **configs = NULL;
    SurfaceFinishMeasurements **measurements = NULL;
    int numConfigs = 0, numMeasurements = 0;

    allocateSurfaceFinishConfigs(&configs, &numConfigs);
    allocateSurfaceFinishMeasurements(&measurements, &numMeasurements);

    for (int i = 0; i < numConfigs; i++) {
        printf("\nEnter details for Configuration %d:\n", i + 1);
        inputSurfaceFinishConfig(configs[i]);
    }

    for (int i = 0; i < numMeasurements; i++) {
        printf("\nEnter surface finish measurement %d:\n", i + 1);
```

```
        inputSurfaceFinishMeasurements(measurements[i]);
    }

    for (int i = 0; i < numConfigs; i++) {
        printSurfaceFinishConfig(configs[i]);
    }

    for (int i = 0; i < numMeasurements; i++) {
        printSurfaceFinishMeasurements(measurements[i]);
    }

    freeSurfaceFinishConfigs(&configs, numConfigs);
    freeSurfaceFinishMeasurements(&measurements, numMeasurements);

    return 0;
}

//10.

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

#define MAX_PROCESSES 100
#define MAX_HEAT_DATA 5

typedef enum {
    IRON_ORE,
    COPPER_ORE,
    GOLD_ORE
} OreType;
```

```
typedef union {
    float ironProperties[3];
    float copperProperties[3];
    float goldProperties[3];
} OreProperties;

typedef struct {
    int processID;
    OreType oreType;
    float temperature;
} SmeltingProcess;

typedef struct {
    OreType oreType;
    OreProperties oreProperties;
} OreData;

void inputSmeltingProcess(SmeltingProcess *process) {
    printf("Enter Process ID: ");
    scanf("%d", &process->processID);
    printf("Enter Ore Type (0: Iron, 1: Copper, 2: Gold): ");
    scanf("%d", &process->oreType);
    printf("Enter Temperature (Celsius): ");
    scanf("%f", &process->temperature);
}

void inputOreProperties(OreData *oreData) {
    printf("Enter Ore Type (0: Iron, 1: Copper, 2: Gold): ");
    scanf("%d", &oreData->oreType);
```

```
    if (oreData->oreType == IRON_ORE) {
        printf("Enter Iron Ore Properties (3 values): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &oreData->oreProperties.ironProperties[i]);
        }
    } else if (oreData->oreType == COPPER_ORE) {
        printf("Enter Copper Ore Properties (3 values): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &oreData->oreProperties.copperProperties[i]);
        }
    } else if (oreData->oreType == GOLD_ORE) {
        printf("Enter Gold Ore Properties (3 values): ");
        for (int i = 0; i < 3; i++) {
            scanf("%f", &oreData->oreProperties.goldProperties[i]);
        }
    }
}

void printSmeltingProcess(const SmeltingProcess *process) {
    printf("\nProcess ID: %d\n", process->processID);
    printf("Ore Type: ");
    if (process->oreType == IRON_ORE) {
        printf("Iron Ore\n");
    } else if (process->oreType == COPPER_ORE) {
        printf("Copper Ore\n");
    } else if (process->oreType == GOLD_ORE) {
        printf("Gold Ore\n");
    }
    printf("Temperature: %.2f°C\n", process->temperature);
}
```

```
void printOreProperties(const OreData *oreData) {
    printf("Ore Type: ");
    if (oreData->oreType == IRON_ORE) {
        printf("Iron Ore\n");
        for (int i = 0; i < 3; i++) {
            printf("Property %d: %.2f\n", i + 1, oreData->oreProperties.ironProperties[i]);
        }
    } else if (oreData->oreType == COPPER_ORE) {
        printf("Copper Ore\n");
        for (int i = 0; i < 3; i++) {
            printf("Property %d: %.2f\n", i + 1, oreData->oreProperties.copperProperties[i]);
        }
    } else if (oreData->oreType == GOLD_ORE) {
        printf("Gold Ore\n");
        for (int i = 0; i < 3; i++) {
            printf("Property %d: %.2f\n", i + 1, oreData->oreProperties.goldProperties[i]);
        }
    }
}

void allocateSmeltingProcesses(SmeltingProcess ***processes, int *numProcesses) {
    printf("Enter the number of smelting processes: ");
    scanf("%d", numProcesses);
    *processes = (SmeltingProcess **)malloc(*numProcesses * sizeof(SmeltingProcess *));
    for (int i = 0; i < *numProcesses; i++) {
        (*processes)[i] = (SmeltingProcess *)malloc(sizeof(SmeltingProcess));
    }
}

void allocateOreData(OreData ***oreData, int *numOreData) {
    printf("Enter the number of ore data sets: ");
```

```
    scanf("%d", numOreData);
    *oreData = (OreData **)malloc(*numOreData * sizeof(OreData *));
    for (int i = 0; i < *numOreData; i++) {
        (*oreData)[i] = (OreData *)malloc(sizeof(OreData));
    }
}

void freeSmeltingProcesses(SmeltingProcess ***processes, int numProcesses) {
    for (int i = 0; i < numProcesses; i++) {
        free((*processes)[i]);
    }
    free(*processes);
}

void freeOreData(OreData ***oreData, int numOreData) {
    for (int i = 0; i < numOreData; i++) {
        free((*oreData)[i]);
    }
    free(*oreData);
}

int main() {
    SmeltingProcess **processes = NULL;
    OreData **oreData = NULL;
    int numProcesses = 0, numOreData = 0;

    allocateSmeltingProcesses(&processes, &numProcesses);
    allocateOreData(&oreData, &numOreData);

    for (int i = 0; i < numProcesses; i++) {
        printf("\nEnter details for Smelting Process %d:\n", i + 1);
```

```
        inputSmeltingProcess(processes[i]);
    }

    for (int i = 0; i < numOreData; i++) {
        printf("\nEnter ore data %d:\n", i + 1);
        inputOreProperties(oreData[i]);
    }

    for (int i = 0; i < numProcesses; i++) {
        printSmeltingProcess(processes[i]);
    }

    for (int i = 0; i < numOreData; i++) {
        printOreProperties(oreData[i]);
    }

    freeSmeltingProcesses(&processes, numProcesses);
    freeOreData(&oreData, numOreData);

    return 0;
}

//11.

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

#define MAX_PLATING_CONFIGS 100
#define MAX_PARAMETERS 5
```

```
typedef struct {
    char ionType[50];
    float charge;
    float concentration;
} MetalIon;

typedef struct {
    float currentDensity;
    float voltage;
    float time;
} PlatingParameters;

void inputMetalIon(MetalIon *ion) {
    printf("Enter Ion Type: ");
    getchar();
    fgets(ion->ionType, sizeof(ion->ionType), stdin);
    ion->ionType[strcspn(ion->ionType, "\n")] = 0;
    printf("Enter Charge (in Coulombs): ");
    scanf("%f", &ion->charge);
    printf("Enter Concentration (mol/L): ");
    scanf("%f", &ion->concentration);
}

void inputPlatingParameters(PlatingParameters *params) {
    printf("Enter Current Density (A/m^2): ");
    scanf("%f", &params->currentDensity);
    printf("Enter Voltage (V): ");
    scanf("%f", &params->voltage);
    printf("Enter Time (s): ");
    scanf("%f", &params->time);
}
```

```
void printMetalIon(const MetalIon *ion) {
    printf("\nIon Type: %s\n", ion->ionType);
    printf("Charge: %.2f C\n", ion->charge);
    printf("Concentration: %.2f mol/L\n", ion->concentration);
}

void printPlatingParameters(const PlatingParameters *params) {
    printf("\nCurrent Density: %.2f A/m^2\n", params->currentDensity);
    printf("Voltage: %.2f V\n", params->voltage);
    printf("Time: %.2f s\n", params->time);
}

void allocatePlatingConfigs(MetalIon ***ions, PlatingParameters ***params, int *numConfigs) {
    printf("Enter the number of electroplating configurations: ");
    scanf("%d", numConfigs);
    *ions = (MetalIon **)malloc(*numConfigs * sizeof(MetalIon *));
    *params = (PlatingParameters **)malloc(*numConfigs * sizeof(PlatingParameters *));
    for (int i = 0; i < *numConfigs; i++) {
        (*ions)[i] = (MetalIon *)malloc(sizeof(MetalIon));
        (*params)[i] = (PlatingParameters *)malloc(sizeof(PlatingParameters));
    }
}

void freePlatingConfigs(MetalIon ***ions, PlatingParameters ***params, int numConfigs) {
    for (int i = 0; i < numConfigs; i++) {
        free((*ions)[i]);
        free((*params)[i]);
    }
    free(*ions);
    free(*params);
```

```
}

int main() {
    MetalIon **ions = NULL;
    PlatingParameters **params = NULL;
    int numConfigs = 0;

    allocatePlatingConfigs(&ions, &params, &numConfigs);

    for (int i = 0; i < numConfigs; i++) {
        printf("\nEnter details for Electroplating Configuration %d:\n", i + 1);
        inputMetalIon(ions[i]);
        inputPlatingParameters(params[i]);
    }

    for (int i = 0; i < numConfigs; i++) {
        printMetalIon(ions[i]);
        printPlatingParameters(params[i]);
    }

    freePlatingConfigs(&ions, &params, numConfigs);

    return 0;
}
//12.

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

#define MAX_DEFECTS 100
```

```
typedef enum {
    SHRINKAGE,
    POROSITY
} DefectType;

typedef union {
    float shrinkageRatio;
    float porosityVolume;
} DefectDetails;

typedef struct {
    int castingID;
    char material[50];
    float length;
    float width;
    float height;
} CastingDetails;

typedef struct {
    DefectType defectType;
    DefectDetails defectDetails;
} DefectData;

void inputCastingDetails(CastingDetails *casting) {
    printf("Enter Casting ID: ");
    scanf("%d", &casting->castingID);
    printf("Enter Material: ");
    getchar();
    fgets(casting->material, sizeof(casting->material), stdin);
    casting->material[strcspn(casting->material, "\n")] = 0;
```

```
    printf("Enter Length (cm): ");
    scanf("%f", &casting->length);
    printf("Enter Width (cm): ");
    scanf("%f", &casting->width);
    printf("Enter Height (cm): ");
    scanf("%f", &casting->height);
}

void inputDefectData(DefectData *defect) {
    printf("Enter Defect Type (0: Shrinkage, 1: Porosity): ");
    scanf("%d", &defect->defectType);

    if (defect->defectType == SHRINKAGE) {
        printf("Enter Shrinkage Ratio: ");
        scanf("%f", &defect->defectDetails.shrinkageRatio);
    } else if (defect->defectType == POROSITY) {
        printf("Enter Porosity Volume: ");
        scanf("%f", &defect->defectDetails.porosityVolume);
    }
}

void printCastingDetails(const CastingDetails *casting) {
    printf("\nCasting ID: %d\n", casting->castingID);
    printf("Material: %s\n", casting->material);
    printf("Dimensions (L x W x H): %.2f x %.2f x %.2f cm\n", casting->length, casting->width, casting->height);
}

void printDefectData(const DefectData *defect) {
    printf("Defect Type: ");
    if (defect->defectType == SHRINKAGE) {
```

```
        printf("Shrinkage\n");
        printf("Shrinkage Ratio: %.2f\n", defect->defectDetails.shrinkageRatio);
    } else if (defect->defectType == POROSITY) {
        printf("Porosity\n");
        printf("Porosity Volume: %.2f cm^3\n", defect->defectDetails.porosityVolume);
    }
}

void allocateDefectData(DefectData ***defects, int *numDefects) {
    printf("Enter the number of defect records: ");
    scanf("%d", numDefects);
    *defects = (DefectData **)malloc(*numDefects * sizeof(DefectData *));
    for (int i = 0; i < *numDefects; i++) {
        (*defects)[i] = (DefectData *)malloc(sizeof(DefectData));
    }
}

void freeDefectData(DefectData ***defects, int numDefects) {
    for (int i = 0; i < numDefects; i++) {
        free((*defects)[i]);
    }
    free(*defects);
}

int main() {
    CastingDetails casting;
    DefectData **defects = NULL;
    int numDefects = 0;

    inputCastingDetails(&casting);
    allocateDefectData(&defects, &numDefects);
```

```
    for (int i = 0; i < numDefects; i++) {
        printf("\nEnter defect data for defect record %d:\n", i + 1);
        inputDefectData(defects[i]);
    }

    printCastingDetails(&casting);
    for (int i = 0; i < numDefects; i++) {
        printDefectData(defects[i]);
    }

    freeDefectData(&defects, numDefects);

    return 0;
}

//13.

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

#define MAX_TESTS 100
#define MAX_EQUIPMENTS 5

typedef struct {
    int sampleID;
    char sampleType[50];
    float length;
    float width;
    float height;
```

```
} SampleDetails;

typedef struct {
    float tensileStrength;
    float hardness;
    float elongation;
} TestResults;

void inputSampleDetails(SampleDetails *sample) {
    printf("Enter Sample ID: ");
    scanf("%d", &sample->sampleID);
    printf("Enter Sample Type: ");
    getchar(); // To consume the newline character left by previous scanf
    fgets(sample->sampleType, sizeof(sample->sampleType), stdin);
    sample->sampleType[strcspn(sample->sampleType, "\n")] = 0; // Remove newline character
    printf("Enter Length (cm): ");
    scanf("%f", &sample->length);
    printf("Enter Width (cm): ");
    scanf("%f", &sample->width);
    printf("Enter Height (cm): ");
    scanf("%f", &sample->height);
}

void inputTestResults(TestResults *test) {
    printf("Enter Tensile Strength (MPa): ");
    scanf("%f", &test->tensileStrength);
    printf("Enter Hardness (HV): ");
    scanf("%f", &test->hardness);
    printf("Enter Elongation (%%): ");
    scanf("%f", &test->elongation);
}
```

```
void printSampleDetails(const SampleDetails *sample) {
    printf("\nSample ID: %d\n", sample->sampleID);
    printf("Sample Type: %s\n", sample->sampleType);
    printf("Dimensions (L x W x H): %.2f x %.2f x %.2f cm\n", sample->length, sample->width, sample->height);
}

void printTestResults(const TestResults *test) {
    printf("Tensile Strength: %.2f MPa\n", test->tensileStrength);
    printf("Hardness: %.2f HV\n", test->hardness);
    printf("Elongation: %.2f %%\n", test->elongation);
}

void allocateTestRecords(TestResults ***testResults, int *numTests) {
    printf("Enter the number of test records: ");
    scanf("%d", numTests);
    *testResults = (TestResults **)malloc(*numTests * sizeof(TestResults *));
    for (int i = 0; i < *numTests; i++) {
        (*testResults)[i] = (TestResults *)malloc(sizeof(TestResults));
    }
}

void freeTestRecords(TestResults ***testResults, int numTests) {
    for (int i = 0; i < numTests; i++) {
        free((*testResults)[i]);
    }
    free(*testResults);
}

int main() {
```

```
    SampleDetails sample;
    TestResults **testResults = NULL;
    int numTests = 0;

    inputSampleDetails(&sample);
    allocateTestRecords(&testResults, &numTests);

    for (int i = 0; i < numTests; i++) {
        printf("\nEnter details for test record %d:\n", i + 1);
        inputTestResults(testResults[i]);
    }

    printSampleDetails(&sample);
    for (int i = 0; i < numTests; i++) {
        printTestResults(testResults[i]);
    }

    freeTestRecords(&testResults, numTests);

    return 0;
}

//14.

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

#define MAX_TESTS 100

typedef enum {
```

```
    ROCKWELL,
    BRINELL
} HardnessScale;

typedef union {
    float rockwellHardness;
    float brinellHardness;
} HardnessValues;

typedef struct {
    int testID;
    char method[50];
    HardnessScale scale;
    HardnessValues hardnessValue;
} HardnessTestData;

void inputHardnessTestData(HardnessTestData *testData) {
    printf("Enter Test ID: ");
    scanf("%d", &testData->testID);
    printf("Enter Test Method: ");
    getchar(); // To consume newline left by previous input
    fgets(testData->method, sizeof(testData->method), stdin);
    testData->method[strcspn(testData->method, "\n")] = 0; // Remove newline character

    printf("Enter Hardness Scale (0: Rockwell, 1: Brinell): ");
    scanf("%d", (int *)&testData->scale);

    if (testData->scale == ROCKWELL) {
        printf("Enter Rockwell Hardness Value: ");
        scanf("%f", &testData->hardnessValue.rockwellHardness);
    } else if (testData->scale == BRINELL) {
```

```
        printf("Enter Brinell Hardness Value: ");
        scanf("%f", &testData->hardnessValue.brinellHardness);
    }
}

void printHardnessTestData(const HardnessTestData *testData) {
    printf("\nTest ID: %d\n", testData->testID);
    printf("Test Method: %s\n", testData->method);

    if (testData->scale == ROCKWELL) {
        printf("Hardness Scale: Rockwell\n");
        printf("Rockwell Hardness: %.2f\n", testData->hardnessValue.rockwellHardness);
    } else if (testData->scale == BRINELL) {
        printf("Hardness Scale: Brinell\n");
        printf("Brinell Hardness: %.2f\n", testData->hardnessValue.brinellHardness);
    }
}

void allocateHardnessTests(HardnessTestData ***tests, int *numTests) {
    printf("Enter the number of hardness test records: ");
    scanf("%d", numTests);
    *tests = (HardnessTestData **)malloc(*numTests * sizeof(HardnessTestData *));
    for (int i = 0; i < *numTests; i++) {
        (*tests)[i] = (HardnessTestData *)malloc(sizeof(HardnessTestData));
    }
}

void freeHardnessTests(HardnessTestData ***tests, int numTests) {
    for (int i = 0; i < numTests; i++) {
        free((*tests)[i]);
    }
```

```
    free(*tests);
}

int main() {
    HardnessTestData **tests = NULL;
    int numTests = 0;

    allocateHardnessTests(&tests, &numTests);

    for (int i = 0; i < numTests; i++) {
        printf("\nEnter data for hardness test record %d:\n", i + 1);
        inputHardnessTestData(tests[i]);
    }

    for (int i = 0; i < numTests; i++) {
        printHardnessTestData(tests[i]);
    }

    freeHardnessTests(&tests, numTests);

    return 0;
}

//15.

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

#define MAX_PARTICLE_SIZES 100
```

```
typedef union {
    float flowability;
    float compressibility;
} PowderProperties;

typedef struct {
    int materialID;
    char materialType[50];
    float density;  // Density in g/cm^3
} MaterialDetails;

typedef struct {
    float particleSize;  // Particle size in micrometers
} ParticleSizeData;

void inputMaterialDetails(MaterialDetails *material) {
    printf("Enter Material ID: ");
    scanf("%d", &material->materialID);
    printf("Enter Material Type: ");
    getchar();  // To consume the newline left by previous input
    fgets(material->materialType, sizeof(material->materialType), stdin);
    material->materialType[strcspn(material->materialType, "\n")] = 0;  // Remove newline character
    printf("Enter Density (g/cm^3): ");
    scanf("%f", &material->density);
}

void inputPowderProperties(PowderProperties *properties) {
    int choice;
    printf("Select Powder Property (0: Flowability, 1: Compressibility): ");
    scanf("%d", &choice);
```

```
    if (choice == 0) {
        printf("Enter Flowability: ");
        scanf("%f", &properties->flowability);
    } else if (choice == 1) {
        printf("Enter Compressibility: ");
        scanf("%f", &properties->compressibility);
    }
}

void inputParticleSizeData(ParticleSizeData *particleData) {
    printf("Enter Particle Size (micrometers): ");
    scanf("%f", &particleData->particleSize);
}

void printMaterialDetails(const MaterialDetails *material) {
    printf("\nMaterial ID: %d\n", material->materialID);
    printf("Material Type: %s\n", material->materialType);
    printf("Density: %.2f g/cm^3\n", material->density);
}

void printPowderProperties(const PowderProperties *properties) {
    printf("Powder Property: ");
    if (properties->flowability != 0) {
        printf("Flowability: %.2f\n", properties->flowability);
    } else {
        printf("Compressibility: %.2f\n", properties->compressibility);
    }
}

void printParticleSizeData(const ParticleSizeData *particleData) {
    printf("Particle Size: %.2f micrometers\n", particleData->particleSize);
```

```
}

void allocatePowderData(ParticleSizeData ***particleSizes, int *numSizes) {
    printf("Enter the number of particle sizes: ");
    scanf("%d", numSizes);
    *particleSizes = (ParticleSizeData **)malloc(*numSizes * sizeof(ParticleSizeData *));
    for (int i = 0; i < *numSizes; i++) {
        (*particleSizes)[i] = (ParticleSizeData *)malloc(sizeof(ParticleSizeData));
    }
}

void freePowderData(ParticleSizeData ***particleSizes, int numSizes) {
    for (int i = 0; i < numSizes; i++) {
        free((*particleSizes)[i]);
    }
    free(*particleSizes);
}

int main() {
    MaterialDetails material;
    PowderProperties powderProperties;
    ParticleSizeData **particleSizes = NULL;
    int numParticleSizes = 0;

    inputMaterialDetails(&material);
    inputPowderProperties(&powderProperties);
    allocatePowderData(&particleSizes, &numParticleSizes);

    for (int i = 0; i < numParticleSizes; i++) {
        printf("\nEnter data for particle size record %d:\n", i + 1);
        inputParticleSizeData(particleSizes[i]);
```

```
    }

    printMaterialDetails(&material);
    printPowderProperties(&powderProperties);

    printf("\nParticle  Size Distribution:\n");
    for (int i = 0; i < numParticleSizes;  i++) {
        printParticleSizeData(particleSizes[i]);
    }

    freePowderData(&particleSizes,  numParticleSizes);

    return 0;
}



//12.

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

#define MAX_DEFECTS 100

typedef enum {
    SHRINKAGE,
    POROSITY
} DefectType;

typedef union {
    float shrinkageRatio;
```

```
    float porosityVolume;
} DefectDetails;

typedef struct {
    int castingID;
    char material[50];
    float length;
    float width;
    float height;
} CastingDetails;

typedef struct {
    DefectType defectType;
    DefectDetails defectDetails;
} DefectData;

void inputCastingDetails(CastingDetails *casting) {
    printf("Enter Casting ID: ");
    scanf("%d", &casting->castingID);
    printf("Enter Material: ");
    getchar();
    fgets(casting->material, sizeof(casting->material), stdin);
    casting->material[strcspn(casting->material, "\n")] = 0;
    printf("Enter Length (cm): ");
    scanf("%f", &casting->length);
    printf("Enter Width (cm): ");
    scanf("%f", &casting->width);
    printf("Enter Height (cm): ");
    scanf("%f", &casting->height);
}
```

```
void inputDefectData(DefectData *defect) {
    printf("Enter Defect Type (0: Shrinkage, 1: Porosity): ");
    scanf("%d", &defect->defectType);

    if (defect->defectType == SHRINKAGE) {
        printf("Enter Shrinkage Ratio: ");
        scanf("%f", &defect->defectDetails.shrinkageRatio);
    } else if (defect->defectType == POROSITY) {
        printf("Enter Porosity Volume: ");
        scanf("%f", &defect->defectDetails.porosityVolume);
    }
}

void printCastingDetails(const CastingDetails *casting) {
    printf("\nCasting ID: %d\n", casting->castingID);
    printf("Material: %s\n", casting->material);
    printf("Dimensions (L x W x H): %.2f x %.2f x %.2f cm\n", casting->length, casting->width, casting->height);
}

void printDefectData(const DefectData *defect) {
    printf("Defect Type: ");
    if (defect->defectType == SHRINKAGE) {
        printf("Shrinkage\n");
        printf("Shrinkage Ratio: %.2f\n", defect->defectDetails.shrinkageRatio);
    } else if (defect->defectType == POROSITY) {
        printf("Porosity\n");
        printf("Porosity Volume: %.2f cm^3\n", defect->defectDetails.porosityVolume);
    }
}
```

```
void allocateDefectData(DefectData ***defects, int *numDefects) {
    printf("Enter the number of defect records: ");
    scanf("%d", numDefects);
    *defects = (DefectData **)malloc(*numDefects * sizeof(DefectData *));
    for (int i = 0; i < *numDefects; i++) {
        (*defects)[i] = (DefectData *)malloc(sizeof(DefectData));
    }
}

void freeDefectData(DefectData ***defects, int numDefects) {
    for (int i = 0; i < numDefects; i++) {
        free((*defects)[i]);
    }
    free(*defects);
}

int main() {
    CastingDetails casting;
    DefectData **defects = NULL;
    int numDefects = 0;

    inputCastingDetails(&casting);
    allocateDefectData(&defects, &numDefects);

    for (int i = 0; i < numDefects; i++) {
        printf("\nEnter defect data for defect record %d:\n", i + 1);
        inputDefectData(defects[i]);
    }

    printCastingDetails(&casting);
    for (int i = 0; i < numDefects; i++) {
```

```
        printDefectData(defects[i]);
    }

    freeDefectData(&defects, numDefects);

    return 0;
}

//13.

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

#define MAX_TESTS 100
#define MAX_EQUIPMENTS 5

typedef struct {
    int sampleID;
    char sampleType[50];
    float length;
    float width;
    float height;
} SampleDetails;

typedef struct {
    float tensileStrength;
    float hardness;
    float elongation;
} TestResults;
```

```
void inputSampleDetails(SampleDetails *sample) {
    printf("Enter Sample ID: ");
    scanf("%d", &sample->sampleID);
    printf("Enter Sample Type: ");
    getchar(); // To consume the newline character left by previous scanf
    fgets(sample->sampleType, sizeof(sample->sampleType), stdin);
    sample->sampleType[strcspn(sample->sampleType, "\n")] = 0; // Remove newline character
    printf("Enter Length (cm): ");
    scanf("%f", &sample->length);
    printf("Enter Width (cm): ");
    scanf("%f", &sample->width);
    printf("Enter Height (cm): ");
    scanf("%f", &sample->height);
}

void inputTestResults(TestResults *test) {
    printf("Enter Tensile Strength (MPa): ");
    scanf("%f", &test->tensileStrength);
    printf("Enter Hardness (HV): ");
    scanf("%f", &test->hardness);
    printf("Enter Elongation (%%): ");
    scanf("%f", &test->elongation);
}

void printSampleDetails(const SampleDetails *sample) {
    printf("\nSample ID: %d\n", sample->sampleID);
    printf("Sample Type: %s\n", sample->sampleType);
    printf("Dimensions (L x W x H): %.2f x %.2f x %.2f cm\n", sample->length, sample->width, sample-
>height);
}
```

```
void printTestResults(const TestResults *test) {
    printf("Tensile Strength: %.2f MPa\n", test->tensileStrength);
    printf("Hardness: %.2f HV\n", test->hardness);
    printf("Elongation: %.2f %%\n", test->elongation);
}

void allocateTestRecords(TestResults ***testResults, int *numTests) {
    printf("Enter the number of test records: ");
    scanf("%d", numTests);
    *testResults = (TestResults **)malloc(*numTests * sizeof(TestResults *));
    for (int i = 0; i < *numTests; i++) {
        (*testResults)[i] = (TestResults *)malloc(sizeof(TestResults));
    }
}

void freeTestRecords(TestResults ***testResults, int numTests) {
    for (int i = 0; i < numTests; i++) {
        free((*testResults)[i]);
    }
    free(*testResults);
}

int main() {
    SampleDetails sample;
    TestResults **testResults = NULL;
    int numTests = 0;

    inputSampleDetails(&sample);
    allocateTestRecords(&testResults, &numTests);

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

```
        printf("\nEnter  details for test record %d:\n", i + 1);
        inputTestResults(testResults[i]);
    }

    printSampleDetails(&sample);
    for (int i = 0; i < numTests;  i++) {
        printTestResults(testResults[i]);
    }

    freeTestRecords(&testResults,  numTests);

    return 0;
}

//14.

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

#define MAX_TESTS 100

typedef enum {
    ROCKWELL,
    BRINELL
} HardnessScale;

typedef union {
    float rockwellHardness;
    float brinellHardness;
} HardnessValues;
```

```
typedef struct {
    int testID;
    char method[50];
    HardnessScale scale;
    HardnessValues hardnessValue;
} HardnessTestData;

void inputHardnessTestData(HardnessTestData *testData) {
    printf("Enter Test ID: ");
    scanf("%d", &testData->testID);
    printf("Enter Test Method: ");
    getchar(); // To consume newline left by previous input
    fgets(testData->method, sizeof(testData->method), stdin);
    testData->method[strcspn(testData->method, "\n")] = 0; // Remove newline character

    printf("Enter Hardness Scale (0: Rockwell, 1: Brinell): ");
    scanf("%d", (int *)&testData->scale);

    if (testData->scale == ROCKWELL) {
        printf("Enter Rockwell Hardness Value: ");
        scanf("%f", &testData->hardnessValue.rockwellHardness);
    } else if (testData->scale == BRINELL) {
        printf("Enter Brinell Hardness Value: ");
        scanf("%f", &testData->hardnessValue.brinellHardness);
    }
}

void printHardnessTestData(const HardnessTestData *testData) {
    printf("\nTest ID: %d\n", testData->testID);
    printf("Test Method: %s\n", testData->method);
```

```
    if (testData->scale == ROCKWELL) {
        printf("Hardness Scale: Rockwell\n");
        printf("Rockwell Hardness: %.2f\n", testData->hardnessValue.rockwellHardness);
    } else if (testData->scale == BRINELL) {
        printf("Hardness Scale: Brinell\n");
        printf("Brinell Hardness: %.2f\n", testData->hardnessValue.brinellHardness);
    }
}

void allocateHardnessTests(HardnessTestData ***tests, int *numTests) {
    printf("Enter the number of hardness test records: ");
    scanf("%d", numTests);
    *tests = (HardnessTestData **)malloc(*numTests * sizeof(HardnessTestData *));
    for (int i = 0; i < *numTests; i++) {
        (*tests)[i] = (HardnessTestData *)malloc(sizeof(HardnessTestData));
    }
}

void freeHardnessTests(HardnessTestData ***tests, int numTests) {
    for (int i = 0; i < numTests; i++) {
        free((*tests)[i]);
    }
    free(*tests);
}

int main() {
    HardnessTestData **tests = NULL;
    int numTests = 0;

    allocateHardnessTests(&tests, &numTests);
```

```
    for (int i = 0; i < numTests; i++) {
        printf("\nEnter data for hardness test record %d:\n", i + 1);
        inputHardnessTestData(tests[i]);
    }

    for (int i = 0; i < numTests; i++) {
        printHardnessTestData(tests[i]);
    }

    freeHardnessTests(&tests, numTests);

    return 0;
}

//15.

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

#define MAX_PARTICLE_SIZES 100

typedef union {
    float flowability;
    float compressibility;
} PowderProperties;

typedef struct {
    int materialID;
    char materialType[50];
```

```
    float density;  // Density in g/cm^3
} MaterialDetails;

typedef struct {
    float particleSize;
} ParticleSizeData;

void inputMaterialDetails(MaterialDetails *material) {
    printf("Enter Material ID: ");
    scanf("%d", &material->materialID);
    printf("Enter Material Type: ");
    getchar();  // To consume the newline left by previous input
    fgets(material->materialType, sizeof(material->materialType), stdin);
    material->materialType[strcspn(material->materialType, "\n")] = 0;  // Remove newline character
    printf("Enter Density (g/cm^3): ");
    scanf("%f", &material->density);
}

void inputPowderProperties(PowderProperties *properties) {
    int choice;
    printf("Select Powder Property (0: Flowability, 1: Compressibility): ");
    scanf("%d", &choice);

    if (choice == 0) {
        printf("Enter Flowability: ");
        scanf("%f", &properties->flowability);
    } else if (choice == 1) {
        printf("Enter Compressibility: ");
        scanf("%f", &properties->compressibility);
    }
}
```

```
void inputParticleSizeData(ParticleSizeData *particleData) {
    printf("Enter Particle Size (micrometers): ");
    scanf("%f", &particleData->particleSize);
}

void printMaterialDetails(const MaterialDetails *material) {
    printf("\nMaterial ID: %d\n", material->materialID);
    printf("Material Type: %s\n", material->materialType);
    printf("Density: %.2f g/cm^3\n", material->density);
}

void printPowderProperties(const PowderProperties *properties) {
    printf("Powder Property: ");
    if (properties->flowability != 0) {
        printf("Flowability: %.2f\n", properties->flowability);
    } else {
        printf("Compressibility: %.2f\n", properties->compressibility);
    }
}

void printParticleSizeData(const ParticleSizeData *particleData) {
    printf("Particle Size: %.2f micrometers\n", particleData->particleSize);
}

void allocatePowderData(ParticleSizeData ***particleSizes, int *numSizes) {
    printf("Enter the number of particle sizes: ");
    scanf("%d", numSizes);
    *particleSizes = (ParticleSizeData **)malloc(*numSizes * sizeof(ParticleSizeData *));
    for (int i = 0; i < *numSizes; i++) {
        (*particleSizes)[i] = (ParticleSizeData *)malloc(sizeof(ParticleSizeData));
```

```
    }
}

void freePowderData(ParticleSizeData ***particleSizes, int numSizes) {
    for (int i = 0; i < numSizes; i++) {
        free((*particleSizes)[i]);
    }
    free(*particleSizes);
}

int main() {
    MaterialDetails material;
    PowderProperties powderProperties;
    ParticleSizeData **particleSizes = NULL;
    int numParticleSizes = 0;

    inputMaterialDetails(&material);
    inputPowderProperties(&powderProperties);
    allocatePowderData(&particleSizes, &numParticleSizes);

    for (int i = 0; i < numParticleSizes; i++) {
        printf("\nEnter data for particle size record %d:\n", i + 1);
        inputParticleSizeData(particleSizes[i]);
    }

    printMaterialDetails(&material);
    printPowderProperties(&powderProperties);

    printf("\nParticle Size Distribution:\n");
    for (int i = 0; i < numParticleSizes; i++) {
        printParticleSizeData(particleSizes[i]);
```

```
    }

    freePowderData(&particleSizes, numParticleSizes);

    return 0;
}

//16.

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

#define MAX_IMPURITY_LEVELS 100
#define MAX_METHOD_LENGTH 50

typedef struct {
    int materialID;
    char materialType[50];
    char recyclingMethod[MAX_METHOD_LENGTH];
} MaterialDetails;

void inputMaterialDetails(MaterialDetails *material) {
    printf("Enter Material ID: ");
    scanf("%d", &material->materialID);
    printf("Enter Material Type: ");
    getchar(); // To consume the newline character left by previous input
    fgets(material->materialType, sizeof(material->materialType), stdin);
    material->materialType[strcspn(material->materialType, "\n")] = 0; // Remove newline character
    printf("Enter Recycling Method: ");
    fgets(material->recyclingMethod, sizeof(material->recyclingMethod), stdin);
```

```
    material->recyclingMethod[strcspn(material->recyclingMethod, "\n")] = 0; // Remove newline character
}

void inputImpurityLevels(float *impurities, int numImpurities) {
    for (int i = 0; i < numImpurities; i++) {
        printf("Enter impurity level %d (percentage): ", i + 1);
        scanf("%f", &impurities[i]);
    }
}

void printMaterialDetails(const MaterialDetails *material) {
    printf("\nMaterial ID: %d\n", material->materialID);
    printf("Material Type: %s\n", material->materialType);
    printf("Recycling Method: %s\n", material->recyclingMethod);
}

void printImpurityLevels(float *impurities, int numImpurities) {
    printf("\nImpurity Levels:\n");
    for (int i = 0; i < numImpurities; i++) {
        printf("Impurity %d: %.2f%%\n", i + 1, impurities[i]);
    }
}

void allocateRecyclingRecords(MaterialDetails ***materials, int *numMaterials) {
    printf("Enter the number of recycling records: ");
    scanf("%d", numMaterials);
    *materials = (MaterialDetails **)malloc(*numMaterials * sizeof(MaterialDetails *));
    for (int i = 0; i < *numMaterials; i++) {
        (*materials)[i] = (MaterialDetails *)malloc(sizeof(MaterialDetails));
    }
```

```
}

void freeRecyclingRecords(MaterialDetails ***materials, int numMaterials) {
    for (int i = 0; i < numMaterials; i++) {
        free((*materials)[i]);
    }
    free(*materials);
}

int main() {
    MaterialDetails **materials = NULL;
    int numMaterials = 0;
    int numImpurities;

    allocateRecyclingRecords(&materials, &numMaterials);

    for (int i = 0; i < numMaterials; i++) {
        printf("\nEnter details for recycling record %d:\n", i + 1);
        inputMaterialDetails(materials[i]);

        printf("Enter number of impurities for material %d: ", i + 1);
        scanf("%d", &numImpurities);
        float *impurities = (float *)malloc(numImpurities * sizeof(float));

        inputImpurityLevels(impurities, numImpurities);
        printMaterialDetails(materials[i]);
        printImpurityLevels(impurities, numImpurities);

        free(impurities);
    }
```

```
    freeRecyclingRecords(&materials, numMaterials);

    return 0;
}

//17.

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

#define MAX_OUTPUT_DATA 100

typedef struct {
    int millID;
    float rollDiameter;  // Roll diameter in cm
    float speed;         // Mill speed in meters per minute
} MillConfig;

void inputMillConfig(MillConfig *mill) {
    printf("Enter Mill ID: ");
    scanf("%d", &mill->millID);
    printf("Enter Roll Diameter (cm): ");
    scanf("%f", &mill->rollDiameter);
    printf("Enter Mill Speed (m/min): ");
    scanf("%f", &mill->speed);
}

void inputOutputData(float *outputData, int numOutputs) {
    for (int i = 0; i < numOutputs; i++) {
        printf("Enter output data for record %d: ", i + 1);
```

```
        scanf("%f", &outputData[i]);
    }
}

void printMillConfig(const MillConfig *mill) {
    printf("\nMill ID: %d\n", mill->millID);
    printf("Roll Diameter: %.2f cm\n", mill->rollDiameter);
    printf("Mill Speed: %.2f m/min\n", mill->speed);
}

void printOutputData(float *outputData, int numOutputs) {
    printf("\nOutput Data:\n");
    for (int i = 0; i < numOutputs; i++) {
        printf("Record %d: %.2f\n", i + 1, outputData[i]);
    }
}

void allocateRollingMillRecords(MillConfig ***mills, int *numMills) {
    printf("Enter the number of rolling mill records: ");
    scanf("%d", numMills);
    *mills = (MillConfig **)malloc(*numMills * sizeof(MillConfig *));
    for (int i = 0; i < *numMills; i++) {
        (*mills)[i] = (MillConfig *)malloc(sizeof(MillConfig));
    }
}

void freeRollingMillRecords(MillConfig ***mills, int numMills) {
    for (int i = 0; i < numMills; i++) {
        free((*mills)[i]);
    }
    free(*mills);
```

```
}

int main() {
    MillConfig **mills = NULL;
    int numMills = 0;
    int numOutputs;

    allocateRollingMillRecords(&mills, &numMills);

    for (int i = 0; i < numMills; i++) {
        printf("\nEnter data for rolling mill record %d:\n", i + 1);
        inputMillConfig(mills[i]);

        printf("Enter the number of output data records for mill %d: ", i + 1);
        scanf("%d", &numOutputs);

        float *outputData = (float *)malloc(numOutputs * sizeof(float));
        inputOutputData(outputData, numOutputs);
        printMillConfig(mills[i]);
        printOutputData(outputData, numOutputs);

        free(outputData);
    }

    freeRollingMillRecords(&mills, numMills);

    return 0;
}

//18.
```

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

#define MAX_TEMPERATURE_DATA 100

typedef union {
    float coefficient; // Coefficient of thermal expansion
} ExpansionCoefficient;

typedef struct {
    int materialID;
    char materialType[50];
    ExpansionCoefficient coeff; // Thermal expansion coefficient
} MaterialProperties;

void inputMaterialProperties(MaterialProperties *material) {
    printf("Enter Material ID: ");
    scanf("%d", &material->materialID);
    printf("Enter Material Type: ");
    getchar(); // To consume the newline character left by previous input
    fgets(material->materialType, sizeof(material->materialType), stdin);
    material->materialType[strcspn(material->materialType, "\n")] = 0; // Remove newline character
    printf("Enter Coefficient of Thermal Expansion: ");
    scanf("%f", &material->coeff.coefficient);
}

void inputTemperatureData(float *temperatures, int numTemperatures) {
    for (int i = 0; i < numTemperatures; i++) {
        printf("Enter temperature data %d: ", i + 1);
        scanf("%f", &temperatures[i]);
```

```
    }
}

void printMaterialProperties(const MaterialProperties *material) {
    printf("\nMaterial ID: %d\n", material->materialID);
    printf("Material Type: %s\n", material->materialType);
    printf("Thermal Expansion Coefficient: %.2f\n", material->coeff.coefficient);
}

void printTemperatureData(float *temperatures, int numTemperatures) {
    printf("\nTemperature Data:\n");
    for (int i = 0; i < numTemperatures; i++) {
        printf("Temperature %d: %.2f°C\n", i + 1, temperatures[i]);
    }
}

void allocateThermalExpansionRecords(MaterialProperties ***materials, int *numMaterials) {
    printf("Enter the number of materials: ");
    scanf("%d", numMaterials);
    *materials = (MaterialProperties **)malloc(*numMaterials * sizeof(MaterialProperties *));
    for (int i = 0; i < *numMaterials; i++) {
        (*materials)[i] = (MaterialProperties *)malloc(sizeof(MaterialProperties));
    }
}

void freeThermalExpansionRecords(MaterialProperties ***materials, int numMaterials) {
    for (int i = 0; i < numMaterials; i++) {
        free((*materials)[i]);
    }
    free(*materials);
}
```

```
int main() {
    MaterialProperties **materials = NULL;
    int numMaterials = 0;
    int numTemperatures;

    allocateThermalExpansionRecords(&materials, &numMaterials);

    for (int i = 0; i < numMaterials; i++) {
        printf("\nEnter data for material %d:\n", i + 1);
        inputMaterialProperties(materials[i]);

        printf("Enter the number of temperature records for material %d: ", i + 1);
        scanf("%d", &numTemperatures);

        float *temperatures = (float *)malloc(numTemperatures * sizeof(float));
        inputTemperatureData(temperatures, numTemperatures);
        printMaterialProperties(materials[i]);
        printTemperatureData(temperatures, numTemperatures);

        free(temperatures);
    }

    freeThermalExpansionRecords(&materials, numMaterials);

    return 0;
}

//19.

#include <stdio.h>
```

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

#define MAX_TEMPERATURE_DATA 100
#define MAX_METAL_NAME_LENGTH 50

typedef struct {
    int metalID;
    char metalName[MAX_METAL_NAME_LENGTH];
    float meltingPoint;  // Melting point in °C
} MetalDetails;

void inputMetalDetails(MetalDetails *metal) {
    printf("Enter Metal ID: ");
    scanf("%d", &metal->metalID);
    printf("Enter Metal Name: ");
    getchar(); // To consume the newline character left by previous input
    fgets(metal->metalName, sizeof(metal->metalName), stdin);
    metal->metalName[strcspn(metal->metalName, "\n")] = 0; // Remove newline character
    printf("Enter Melting Point (°C): ");
    scanf("%f", &metal->meltingPoint);
}

void inputTemperatureData(float *temperatures, int numTemperatures) {
    for (int i = 0; i < numTemperatures; i++) {
        printf("Enter temperature data %d: ", i + 1);
        scanf("%f", &temperatures[i]);
    }
}

void printMetalDetails(const MetalDetails *metal) {
```

```
    printf("\nMetal ID: %d\n", metal->metalID);
    printf("Metal Name: %s\n", metal->metalName);
    printf("Melting Point: %.2f°C\n", metal->meltingPoint);
}

void printTemperatureData(float *temperatures, int numTemperatures) {
    printf("\nTemperature Data:\n");
    for (int i = 0; i < numTemperatures; i++) {
        printf("Temperature %d: %.2f°C\n", i + 1, temperatures[i]);
    }
}

void allocateMeltingPointRecords(MetalDetails ***metals, int *numMetals) {
    printf("Enter the number of metals: ");
    scanf("%d", numMetals);
    *metals = (MetalDetails **)malloc(*numMetals * sizeof(MetalDetails *));
    for (int i = 0; i < *numMetals; i++) {
        (*metals)[i] = (MetalDetails *)malloc(sizeof(MetalDetails));
    }
}

void freeMeltingPointRecords(MetalDetails ***metals, int numMetals) {
    for (int i = 0; i < numMetals; i++) {
        free((*metals)[i]);
    }
    free(*metals);
}

int main() {
    MetalDetails **metals = NULL;
    int numMetals = 0;
```

```
    int numTemperatures;

    allocateMeltingPointRecords(&metals, &numMetals);

    for (int i = 0; i < numMetals; i++) {
        printf("\nEnter data for metal %d:\n", i + 1);
        inputMetalDetails(metals[i]);

        printf("Enter the number of temperature records for metal %d: ", i + 1);
        scanf("%d", &numTemperatures);

        float *temperatures = (float *)malloc(numTemperatures * sizeof(float));
        inputTemperatureData(temperatures, numTemperatures);
        printMetalDetails(metals[i]);
        printTemperatureData(temperatures, numTemperatures);

        free(temperatures);
    }

    freeMeltingPointRecords(&metals, numMetals);

    return 0;
}

//20.

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

#define MAX_ENERGY_DATA 100
```

```
typedef union {
    float energyConsumption;   // Energy consumption in kWh
    float processDuration;     // Process duration in hours
} ProcessParameters;

typedef struct {
    int processID;
    char oreType[50];
    float efficiency;  // Efficiency as a percentage
    ProcessParameters params;  // Energy consumption or process duration
} SmeltingProcess;

void inputSmeltingProcess(SmeltingProcess *process) {
    printf("Enter Process ID: ");
    scanf("%d", &process->processID);
    printf("Enter Ore Type: ");
    getchar();  // To consume the newline character left by previous input
    fgets(process->oreType, sizeof(process->oreType), stdin);
    process->oreType[strcspn(process->oreType, "\n")] = 0;  // Remove newline character
    printf("Enter Process Efficiency (percentage): ");
    scanf("%f", &process->efficiency);
}

void inputEnergyConsumption(float *energyData, int numRecords) {
    for (int i = 0; i < numRecords; i++) {
        printf("Enter energy consumption data %d: ", i + 1);
        scanf("%f", &energyData[i]);
    }
}
```

```
void printSmeltingProcess(const SmeltingProcess *process) {
    printf("\nProcess ID: %d\n", process->processID);
    printf("Ore Type: %s\n", process->oreType);
    printf("Efficiency: %.2f%%\n", process->efficiency);
}

void printEnergyConsumptionData(float *energyData, int numRecords) {
    printf("\nEnergy Consumption Data:\n");
    for (int i = 0; i < numRecords; i++) {
        printf("Record %d: %.2f kWh\n", i + 1, energyData[i]);
    }
}

void allocateSmeltingEfficiencyRecords(SmeltingProcess ***processes, int *numProcesses) {
    printf("Enter the number of smelting processes: ");
    scanf("%d", numProcesses);
    *processes = (SmeltingProcess **)malloc(*numProcesses * sizeof(SmeltingProcess *));
    for (int i = 0; i < *numProcesses; i++) {
        (*processes)[i] = (SmeltingProcess *)malloc(sizeof(SmeltingProcess));
    }
}

void freeSmeltingEfficiencyRecords(SmeltingProcess ***processes, int numProcesses) {
    for (int i = 0; i < numProcesses; i++) {
        free((*processes)[i]);
    }
    free(*processes);
}

int main() {
    SmeltingProcess **processes = NULL;
```

```
    int numProcesses = 0;
    int numRecords;

    allocateSmeltingEfficiencyRecords(&processes, &numProcesses);

    for (int i = 0; i < numProcesses; i++) {
        printf("\nEnter data for smelting process %d:\n", i + 1);
        inputSmeltingProcess(processes[i]);

        printf("Enter the number of energy consumption records for process %d: ", i + 1);
        scanf("%d", &numRecords);

        float *energyData = (float *)malloc(numRecords * sizeof(float));
        inputEnergyConsumption(energyData, numRecords);
        printSmeltingProcess(processes[i]);
        printEnergyConsumptionData(energyData, numRecords);

        free(energyData);
    }

    freeSmeltingEfficiencyRecords(&processes, numProcesses);

    return 0;
}
```