```c
/*Design a system to analyze alloy compositions using structures for
composition details, arrays for storing multiple samples,
 and unions to represent percentage compositions of different metals.
Specifications:
Structure: Stores sample ID, name, and composition details.
Union: Represents variable percentage compositions of metals.
Array: Stores multiple alloy samples.
const Pointers: Protect composition details.
Double Pointers: Manage dynamic allocation of alloy samples*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Percent
{
    char percent[10];
};
struct Sample
{
    char id[10];
    const char *composition[3];
    union Percent p;
};
void add(struct Sample **s,int *count)
{
    printf("Enter id : ");
    scanf("%s",(*s)[*count].id);
    printf("Elements in alloy : \n");
    for(int i=0;i<3;i++)
    {
        (*s)[*count].composition[i]=(char *)malloc(10*sizeof(char));
        printf("Element %d : ",i+1);
        scanf("%s",(*s)[*count].composition[i]);
    }
    printf("Enter element composition percent : ");
    scanf("%s",(*s)[*count].p.percent);
    (*count)++;

}
void delete(struct Sample **s,int *count,const char *id)
{
```

```
    int index =-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index ==-1)
    {
        printf("Id not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*s)[i]=(*s)[i+1];
    }
    (*count)--;
}
void display(struct Sample *s,int count)
{
    for(int i=0;i<count;i++)
    {
        printf("Sample id : %s | Composition :  ",s[i].id);
        for(int j=0;j<3;j++)
        {
            printf("%s |",s[i].composition[j]);
        }
        printf("Percent : %s\n",s[i].p.percent);
    }
}
void main()
{
    struct Sample *s=(struct Sample *)malloc(10*sizeof(struct Sample));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
```

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

}
/*Develop a program to manage heat treatment processes for metals using
structures for process details, arrays for treatment parameters,
 and strings for process names.
Specifications:
Structure: Holds process ID, temperature, duration, and cooling rate.
Array: Stores treatment parameter sets.
Strings: Process names.
const Pointers: Protect process data.
Double Pointers: Allocate and manage dynamic process data.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct Process
{
    char name[20];
    char id[20];
    const char *temp;
```

```
    const char *dur;
    const char *crate;
};
void add(struct Process **p,int * count)
{
    printf("Enter process id :");
    scanf("%s",(*p)[*count].id);
    printf("Enter process name : ");
    scanf("%s",(*p)[*count].name);
    printf("Enter temperature : ");
    (*p)[*count].temp=(char *)malloc(5*sizeof(char));
    scanf("%s",(*p)[*count].temp);
    printf("Enter Duration (hh:mm:ss) : ");
    (*p)[*count].dur=(char *)malloc(10*sizeof(char));
    scanf("%s",(*p)[*count].dur);
    printf("Enter cooling rate : ");
    (*p)[*count].crate=(char *)malloc(5*sizeof(char));
    scanf("%s",(*p)[*count].crate);
(*count)++;
}
void delete(struct Process **p,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*p)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index ==-1)
    {
        printf("Process not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*p)[i]=(*p)[i+1];
    }
```

```
    (*count)--;
}
void display(struct Process *p,int count)
{
    for(int i=0;i<count;i++)
    {
        printf("Process id : %s | Process name : %s | Temperature : %s |
Duration : %s | Cooling rate :
%s\n",p[i].id,p[i].name,p[i].temp,p[i].dur,p[i].crate);
    }
}
void main()
{
    struct Process *p=(struct Process *)malloc(10*sizeof(struct Process));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&p,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&p,&count,id);
            break;
            case 3: display(p,count);
            break;
            case 4:printf("Exiting ......\n");
            free(p);
            break;
            default :printf("Enter valid option \n");
            break;
        }
```

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

}
/*Create a system to monitor steel quality using structures for test
results, arrays for storing test data,
and unions for variable quality metrics like tensile strength and
hardness.
Specifications:
Structure: Stores test ID, type, and result.
Union: Represents tensile strength, hardness, or elongation.
Array: Test data for multiple samples.
const Pointers: Protect test IDs.
Double Pointers: Manage dynamic test records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Result
{
    float strength;
    char hardness[10];
    float elongation;
};
struct Test
{
    const char * id;
    int type;//0 strength 1 hardness 2 elongation
    union Result r;
};
void add(struct Test **t,int * count)
{
    printf("Enter test id : ");
    (*t)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*t)[*count].id);
    printf("Enter type(0.strength 1.hardness 2.elongation) : ");
    scanf("%d",&(*t)[*count].type);
    if((*t)[*count].type==0)
    {
        printf("Enter strength : ");
        scanf("%f",&(*t)[*count].r.strength);
    }
```

```
    else if((*t)[*count].type==1)
    {
        printf("Enter hardness : ");
        scanf("%s",(*t)[*count].r.hardness);
    }
    else
    {
        printf("Enter elongation :");
        scanf("%f",&(*t)[*count].r.elongation);
    }
    (*count)++;
}
void delete(struct Test **t,int * count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*t)[i].id,id)==0)
        {
            index=i;
            break;

        }
    }
    if(index ==-1)
    {
        printf("Test not found \n");
        return;
    }

    for(int i=index;i<*count;i++)
    {
        (*t)[i]=(*t)[i+1];
    }
    (*count)--;
}
void display(struct Test*t,int count)
{
    for(int i=0;i<count;i++)
    {
```

```
        printf("Test id : %s |",t[i].id);
        if(t[i].type==0)
        {
            printf("Strength : %.2f\n",t[i].r.strength);
        }
        else if(t[i].type==1)
        {
            printf("Hardness : %s\n",t[i].r.hardness);
        }
        else
        printf("Elongation : %.2f\n",t[i].r.elongation);
    }
}
void main()
{
    struct Test *t=(struct Test *)malloc(10*sizeof(struct Test));
    int choice,count=0;
    char id[10];
    do
    {
        printf("1.Add\n");
        printf("2.Remove\n");
        printf("3.Display \n");
        printf("4.Exit\n");
        printf("Enter option : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: add(&t,&count);
            break;
            case 2:printf("Enter id to remmove : ");
            scanf("%s",id);
            delete(&t,&count,id);
            break;
            case 3: display(t,count);
            break;
            case 4:printf("Exiting ......\n");
            free(t);
            break;
            default :printf("Enter valid option \n");
```

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

}
/*Develop a program to analyze metal fatigue using arrays for stress cycle
data, structures for material details, and strings for material names.
Specifications:
Structure: Contains material ID, name, and endurance limit.
Array: Stress cycle data.
Strings: Material names.
const Pointers: Protect material details.
Double Pointers: Allocate dynamic material test data.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct Material
{
    const char *id;
    const char *name;
    const char *limit;
};
void add(struct Material **m,int *count)
{
    printf("Enter material id : ");
    (*m)[*count].id=(char *)malloc(10*sizeof(char));
    scanf("%s",(*m)[*count].id);
    printf("Enter name : ");
    (*m)[*count].name=(char *)malloc(20*sizeof(char));
    scanf("%s",(*m)[*count].name);
    printf("Enter endurance limit : ");
    (*m)[*count].limit=(char *)malloc(8*sizeof(char));
    scanf("%s",(*m)[*count].limit);
    (*count)++;
}
void delete(struct Material **m,int *count ,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
```

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

printf("------------------------------------------------------------------
--------\n");
    for(int i=0;i<count;i++)
    {
        printf("Material id : %s | Name : %s | Limit : %s
\n",m[i].id,m[i].name,m[i].limit);

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

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

}
/*Create a system for managing foundry operations using arrays for
equipment data, structures for casting details,
and unions for variable mold properties.
Specifications:
Structure: Stores casting ID, weight, and material.
Union: Represents mold properties (dimensions or thermal conductivity).
Array: Equipment data.
const Pointers: Protect equipment details.
Double Pointers: Dynamic allocation of casting records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Property
{
    char dim[14];
```

```
    float tc;
};
struct Casting
{
    char id[10];
    float weight;
    char* material;
    int pr;//0 dimension 1 conductivity
    union Property p;
};
void add(struct Casting **c,int *count)
{
    printf("Enter casting id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter weight : ");
    scanf("%f",&(*c)[*count].weight);
    printf("Enter material : ");
    (*c)[*count].material=(char *)malloc(20*sizeof(char));
    scanf("%s",(*c)[*count].material);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].pr);
    if((*c)[*count].pr==0)
    {
        printf("Enter dimensions : ");
        scanf("%s",(*c)[*count].p.dim);
    }
    else
    {
        printf("Enter thermal conductivity : ");
        scanf("%f",&(*c)[*count].p.tc);
    }
(*count)++;
}
void delete(struct Casting **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Casting id : %s | Weight : %f | Material : %s |
",c[i].id,c[i].weight,c[i].material);
        if(c[i].pr==0)
        {
            printf("Dimension : %s\n",c[i].p.dim);
        }
        else if(c[i].pr==1)
        {
            printf("Thermal Conductivity : %s\n",c[i].p.tc);
        }

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

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

}
/*Develop a system for metal purity analysis using structures for sample
data, arrays for impurity percentages, and unions for variable impurity
types.
Specifications:
Structure: Contains sample ID, type, and purity.
Union: Represents impurity type (trace elements or oxides).
Array: Impurity percentages.
const Pointers: Protect purity data.
Double Pointers: Manage dynamic impurity records.*/
#include<stdio.h>
```

```c
#include<string.h>
#include<stdlib.h>
union Type
{
    char elements[20];
    char oxides[20];
};
struct Sample
{
    char* id;
    int type; //0.elements 1.oxides
    float percent;
    union Type t;

};
void add(struct Sample **s,int *count)
{
    printf("Enter sampleid : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter type : ");
    scanf("%d",&(*s)[*count].type);
     if((*s)[*count].type==0)
    {
        printf("Enter elements  : ");
        getchar();
        fgets((*s)[*count].t.elements,20,stdin);
        (*s)[*count].t.elements[strcspn((*s)[*count].t.elements, "\n")] =
'\0';

    }
    else
    {
        printf("Enter oxides: ");
         getchar();
        fgets((*s)[*count].t.oxides,20,stdin);
       (*s)[*count].t.oxides[strcspn((*s)[*count].t.oxides, "\n")] = '\0';
    }
    printf("Enter percent : ");
    scanf("%f",&(*s)[*count].percent);
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Sample id : %s | percent : %.2f | ",s[i].id,s[i].percent);
        if(s[i].type==0)
        {
            printf("Element : %s\n",s[i].t.elements);
        }
        else if(s[i].type==1)
        {
            printf("Oxides : %s\n",s[i].t.oxides);
```

```
        }

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

}
/*Create a program to track corrosion tests using structures for test
details, arrays for test results, and strings for test conditions.
```

```c
/*Specifications:
Structure: Holds test ID, duration, and environment.
Array: Test results.
Strings: Test conditions.
const Pointers: Protect test configurations.
Double Pointers: Dynamic allocation of test records.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct Test
{
    char id[10];
    char dur[10];
    char *en[3];
};
void add(struct Test **t,int *count)
{
    printf("Enter Test id : ");
    scanf("%s",(*t)[*count].id);
    printf("Enter duration : ");
    scanf("%s",&(*t)[*count].dur);
        (*t)[*count].en[0]=(char *)malloc(5 * sizeof(char));
        printf("Enter Pressure : ");
        getchar();
        scanf("%s",(*t)[*count].en[0]);
         (*t)[*count].en[1]=(char *)malloc(5 * sizeof(char));
        printf("Enter Temperature : ");
        getchar();
        scanf("%s",(*t)[*count].en[1]);
         (*t)[*count].en[2]=(char *)malloc(5 * sizeof(char));
        printf("Enter Humidity : ");
        getchar();
        scanf("%s",(*t)[*count].en[2]);
(*count)++;
}
void delete(struct Test **t,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Test id : %s | Duration : %s | Pressure : %s | Temperature
: %s | Humidity :
%s\n",t[i].id,t[i].dur,t[i].en[0],t[i].en[1],t[i].en[2]);

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

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

}
/*Develop a program to optimize welding parameters using structures for
parameter sets, arrays for test outcomes, and unions for variable welding
types.
Specifications:
Structure: Stores parameter ID, voltage, current, and speed.
Union: Represents welding types (MIG, TIG, or Arc).
Array: Test outcomes.
const Pointers: Protect parameter configurations.
Double Pointers: Manage dynamic parameter sets.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Type
{
```

```
    char welding[4];
};
struct Test
{
    char id[10];
    float volt;
    float c;
    char speed[10];
    int type;//0 MIG 1 TIG 2 ARC
    union Type t;
};
void add(struct Test **t,int *count)
{
    printf("Enter Test id : ");
    scanf("%s",(*t)[*count].id);
    printf("Enter voltage :");
    scanf("%f",&(*t)[*count].volt);
    printf("Enter current : ");
    scanf("%f",&(*t)[*count].c);
    printf("Enter time taken (HH;MM;SS) : ");
    scanf("%s",(*t)[*count].speed);
    printf("Enter type : ");
    scanf("%d",&(*t)[*count].type);
    if((*t)[*count].type==0)
    strcpy((*t)[*count].t.welding,"MIG");
    else if((*t)[*count].type==1)
    strcpy((*t)[*count].t.welding,"TIG");
    else
    strcpy((*t)[*count].t.welding,"ARC");
(*count)++;
}
void delete(struct Test **t,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*t)[i].id,id)==0)
        {
            index=i;
            break;
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Weilding id : %s | Duration : %s | Voltage : %.2f |
Current : %.2f | Type :
%s\n",t[i].id,t[i].speed,t[i].volt,t[i].c,t[i].t.welding);

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

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

}
/*Design a program to analyze surface finishes using arrays for
measurement data, structures for Surface configurations, and strings for
surface types.
Specifications:
Structure: Holds configuration ID, material, and measurement units.
Array: Surface finish measurements.
Strings: Surface types.
const Pointers: Protect configuration details.
Double Pointers: Allocate and manage measurement data.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Type
{
    char welding[4];
};
struct Surface
{
    char id[10];
```

```
    char type[10];
    char material[20];
    char measurement[20];
};
void add(struct Surface **s,int *count)
{
    printf("Enter Surface id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter surface type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter material : ");
    scanf("%s",(*s)[*count].material);
    printf("Enter measurement : ");
    getchar();
    fgets((*s)[*count].measurement,20,stdin);
(*count)++;
}
void delete(struct Surface **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Weilding not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*s)[i]=(*s)[i+1];
    }
(*count)--;
}
void display(struct Surface *s,int count)
```

```
{
    printf("\nSurface details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Material id : %s | Type : %s | Material : %s | Measurement
: %s\n",s[i].id,s[i].type,s[i].material,s[i].measurement);

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

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

}
/*Create a system to track smelting processes using structures for process
metadata, arrays for heat data, and unions for variable ore properties.
Specifications:
Structure: Holds process ID, ore type, and temperature.
Union: Represents variable ore properties.
Array: Heat data.
const Pointers: Protect process metadata.
Double Pointers: Allocate dynamic process records.*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
union Property
{
    float carbonContent;
    float sulfurContent;
};
struct Process
{
    char id[10];
    char type[10];
    char temperature[20];
    int tp;//0 carbon 1 sulfur
    union Property p;

};
void add(struct Process **p,int *count)
{
    printf("Enter Process id : ");
    scanf("%s",(*p)[*count].id);
    printf("Enter ore type :");
    scanf("%s",(*p)[*count].type);
    printf("Enter temperature : ");
    scanf("%s",(*p)[*count].temperature);
    printf("Enter ore reaction : ");
```

```
    scanf("%d",&(*p)[*count].tp);
    if((*p)[*count].tp==0)
    {
        printf("Enter carbon content percentage : ");
        scanf("%f",&(*p)[*count].p.carbonContent);
    }
    else
    {
        printf("Enter sulphur content percentage : ");
        scanf("%f",&(*p)[*count].p.sulfurContent);
    }

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

```
printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Process id : %s | Type : %s | temperature : %s
|",p[i].id,p[i].type,p[i].temperature);
        if(p[i].tp==0)
        {
            printf("Contains : Carbon | Carboncontent :
%.2f\n",p[i].p.carbonContent);
        }
        else
        {
            printf("Contains : Sulphur | Sulphurcontent :
%.2f\n",p[i].p.sulfurContent);
        }

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

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

}
/*Simulate an electroplating system using structures for metal ions,
arrays for plating parameters, and strings for electrolyte names.
Specifications:
Structure: Stores ion type, charge, and concentration.
Array: Plating parameters.
Strings: Electrolyte names.
const Pointers: Protect ion data.
Double Pointers: Manage dynamic plating configurations.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Plating
{
    char id[10];
    char type[10];
    char charge[10];
    float conc;

};
void add(struct Plating **p,int *count)
{
    printf("Enter Plating id : ");
    scanf("%s",(*p)[*count].id);
    printf("Enter ion type :");
    scanf("%s",(*p)[*count].type);
    printf("Enter tcharge : ");
```

```
    scanf("%s",(*p)[*count].charge);
    printf("Enter concentration : ");
    scanf("%f",&(*p)[*count].conc);
(*count)++;
}
void delete(struct Plating **p,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*p)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Plating not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*p)[i]=(*p)[i+1];
    }
(*count)--;
}
void display(struct Plating *p,int count)
{
    printf("\nPlating details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Plating id : %s | Ion Type : %s | Charge : %s |
Concentration : %.2f\n",p[i].id,p[i].type,p[i].charge,p[i].conc);

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

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

}
/*Design a system to analyze casting defects using arrays for defect data,
structures for casting details, and unions for variable defect types.
Specifications:
Structure: Holds casting ID, material, and dimensions.
Union: Represents defect types (shrinkage or porosity).
Array: Defect data.
```

```c
const Pointers: Protect casting data.
Double Pointers: Dynamic defect record management.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Defect
{
    float shrinkage;
    float porosity;
};
struct Casting
{
    char id[10];
    char material[10];
    int type;//0.shrinkage 1.porosity;
    union Defect d;

};
void add(struct Casting **c,int *count)
{
    printf("Enter Casting id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material :");
    scanf("%s",(*c)[*count].material);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter shrinkage : ");
    scanf("%f",&(*c)[*count].d.shrinkage);
    }
    else
    {
        printf("Enter porosity : ");
    scanf("%f",&(*c)[*count].d.porosity);
    }

(*count)++;
}
void delete(struct Casting **c,int *count,const char *id)
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Casting id : %s | Material : %s |
",c[i].id,c[i].material);
        if(c[i].type==0)
        {
            printf("Shrinkage : %.2f\n",c[i].d.shrinkage);
        }
        else
        {
            printf("Porosity : %.2f\n",c[i].d.porosity);
        }
```

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

}
/*Automate a metallurgical lab using structures for sample details, arrays
for test results, and strings for equipment names.
Specifications:
```

```c
/*Structure: Contains sample ID, type, and dimensions.
Array: Test results.
Strings: Equipment names.
const Pointers: Protect sample details.
Double Pointers: Allocate and manage dynamic test records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct Sample
{
    char id[10];
    char type[10];
    char dim[20];
    char *equ[3];

};
void add(struct Sample **s,int *count)
{
    printf("Enter Sample id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter dimensions : ");
    getchar();
    fgets((*s)[*count].dim,20,stdin);
    for(int i=0;i<3;i++)
    {
        (*s)[*count].equ[i]=(char *)malloc(10*sizeof(char));
        printf("Enter equipment %d : ",i+1);
        scanf("%s",(*s)[*count].equ[i]);
    }

(*count)++;
}
void delete(struct Sample **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
```

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

printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Sample id : %s | Type : %s | Dimensions : %s |
",s[i].id,s[i].type,s[i].dim);
        for(int j=0;j<3;j++)
        {
            printf("Equipment %d : %s |",j+1,s[i].equ[j]);
        }
        printf("\n");

printf("-------------------------------------------------------------------
----------------------\n");
    }
}
void main()
{
    struct Sample *t=(struct Sample *)malloc(10*sizeof(struct Sample));
```

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

}
/*Develop a program to track metal hardness tests using structures for
test data, arrays for hardness values, and unions for variable hardness
scales.
Specifications:
Structure: Stores test ID, method, and result.
Union: Represents variable hardness scales (Rockwell or Brinell).
Array: Hardness values.
const Pointers: Protect test data.
Double Pointers: Dynamic hardness record allocation.*/
#include<stdio.h>
#include<stdlib.h>
```

```c
#include<string.h>
union Scale
{
    float rockwell;
    float brinell;
};
struct Metal
{
    char id[10];
    char method[10];
    int type;//0.rockwell 1.brinell;
    union Scale s;

};
void add(struct Metal **c,int *count)
{
    printf("Enter Test id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter method :");
    scanf("%s",(*c)[*count].method);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter rokwell : ");
    scanf("%f",&(*c)[*count].s.rockwell);
    }
    else
    {
        printf("Enter brinell : ");
    scanf("%f",&(*c)[*count].s.brinell);
    }

(*count)++;
}
void delete(struct Metal **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Test id : %s | method : %s | ",c[i].id,c[i].method);
        if(c[i].type==0)
        {
            printf("Rockwell : %.2f\n",c[i].s.rockwell);
        }
        else
        {
            printf("Porosity : %.2f\n",c[i].s.brinell);
        }

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

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

}
/*Create a program to track powder metallurgy processes using structures
for material details,
arrays for particle size distribution, and unions for variable powder
properties.
Specifications:
Structure: Contains material ID, type, and density.
Union: Represents powder properties.
Array: Particle size distribution data.
const Pointers: Protect material configurations.
```

```c
Double Pointers: Allocate and manage powder data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Property
{
    float p1;
    float p2;
};
struct Powder
{
    char id[10];
    float density;
    int type;//0.p1 1.p2;
    union Property s;

};
void add(struct Powder **c,int *count)
{
    printf("Enter Material id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter density :");
    scanf("%f",&(*c)[*count].density);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter P1 : ");
    scanf("%f",&(*c)[*count].s.p1);
    }
    else
    {
        printf("Enter P2 : ");
    scanf("%f",&(*c)[*count].s.p2);
    }

(*count)++;
}
void delete(struct Powder **c,int *count,const char *id)
{
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Test id : %s | Density : %.2f | ",c[i].id,c[i].density);
        if(c[i].type==0)
        {
            printf("P1 : %.2f\n",c[i].s.p1);
        }
        else
        {
            printf("P2 : %.2f\n",c[i].s.p2);
        }

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

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

}
/*Develop a program to analyze recycled metal data using structures for
material details, arrays for impurity levels, and strings for recycling
methods.
Specifications:
Structure: Holds material ID, type, and recycling method.
Array: Impurity levels.
```

```c
/*Strings: Recycling methods.
const Pointers: Protect material details.
Double Pointers: Allocate dynamic recycling records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct Metal
{
    char id[10];
    char type[10];
    float impurity;
    char *method[3];

};
void add(struct Metal **s,int *count)
{
    printf("Enter Metal id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter impurity : ");
    scanf("%f",&(*s)[*count].impurity);
    for(int i=0;i<3;i++)
    {
        (*s)[*count].method[i]=(char *)malloc(10*sizeof(char));
        printf("Enter method %d : ",i+1);
        scanf("%s",(*s)[*count].method[i]);
    }

(*count)++;
}
void delete(struct Metal **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
```

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

printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Metal id : %s | Type : %s | Impurity : %.2f |
",s[i].id,s[i].type,s[i].impurity);
        for(int j=0;j<3;j++)
        {
            printf("Method %d : %s |",j+1,s[i].method[j]);
        }
        printf("\n");

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

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

}
/*Design a system to track rolling mill performance using structures for
mill configurations, arrays for output data, and strings for material
types.
Specifications:
Structure: Stores mill ID, roll diameter, and speed.
Array: Output data.
Strings: Material types.
const Pointers: Protect mill configurations.
Double Pointers: Manage rolling mill records dynamically.
 */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```c
struct Mill
{
    char id[10];
    char type[10];
    float dia;
    float speed;

};
void add(struct Mill **s,int *count)
{
    printf("Enter Mill id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter material type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter dia : ");
    scanf("%f",&(*s)[*count].dia);
    printf("Enter speed : ");
    scanf("%f",&(*s)[*count].speed);

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

```
    }
(*count)--;
}
void display(struct Mill *s,int count)
{
    printf("\nMill details \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Mill id : %s | Material : %s | Diameter : %.2f | speed :
%.2f\n",s[i].id,s[i].type,s[i].dia,s[i].speed);

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

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

}
/*Create a program to analyze thermal expansion using arrays for
temperature data, structures for material properties,
 and unions for variable coefficients.
Specifications:
Structure: Contains material ID, type, and expansion coefficient.
Union: Represents variable coefficients.
Array: Temperature data.
const Pointers: Protect material properties.
Double Pointers: Dynamic thermal expansion record allocation.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Coef
{
    float real;
    float imag;
};
struct Metal
{
    char id[10];
    float ex;
    int type;//0.real 1.imaginary;
    union Coef s;

};
void add(struct Metal **c,int *count)
{
    printf("Enter Test id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter expansion coefficent :");
```

```
    scanf("%f",&(*c)[*count].ex);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter real value : ");
    scanf("%f",&(*c)[*count].s.real);
    }
    else
    {
        printf("Enter imaginary : ");
    scanf("%f",&(*c)[*count].s.imag);
    }

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

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Test id : %s | Expansive coefficient : %.2f |
",c[i].id,c[i].ex);
        if(c[i].type==0)
        {
            printf("Real : %.2f\n",c[i].s.real);
        }
        else
        {
            printf("Imaginary : %.2f\n",c[i].s.imag);
        }

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

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

}
/*Develop a program to analyze melting points using structures for metal
details, arrays for temperature data, and strings for metal names.
Specifications:
Structure: Stores metal ID, name, and melting point.
Array: Temperature data.
Strings: Metal names.
const Pointers: Protect metal details.
Double Pointers: Allocate dynamic melting point records*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct Metal
{
    char id[10];
    char name[10];
    float mp;

};
void add(struct Metal **s,int *count)
{
    printf("Enter Metal id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter material name :");
    scanf("%s",(*s)[*count].name);
    printf("Enter melting point : ");
    scanf("%f",&(*s)[*count].mp);
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Metal id : %s | Name : %s | Melting point :
%.2f\n",s[i].id,s[i].name,s[i].mp);

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

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

}
/*Design a system to analyze smelting efficiency using structures for
process details, arrays for energy consumption data,
 and unions for variable process parameters.
Specifications:
Structure: Contains process ID, ore type, and efficiency.
Union: Represents process parameters (energy or duration).
Array: Energy consumption data.
const Pointers: Protect process configurations.
```

```c
Double Pointers: Manage smelting efficiency records dynamically*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float energy;
    char duration[20];
};
struct Process
{
    char id[10];
    float efficiency;
    int type;//0.metal 1.silica;
    union Parameter s;

};
void add(struct Process **c,int *count)
{
    printf("Enter process id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter efficiency :");
    scanf("%f",&(*c)[*count].efficiency);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter energy value : ");
        getchar();
    scanf("%f",&(*c)[*count].s.energy);
    }
    else
    {
        printf("Enter duration : ");
        getchar();
    fgets((*c)[*count].s.duration,20,stdin);
    }

(*count)++;
}
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Process id : %s | Efficiency : %.2f |
",c[i].id,c[i].efficiency);
        if(c[i].type==0)
        {
            printf("energy : %.2f\n",c[i].s.energy);
        }
        else
        {
            printf("Duration : %s\n",c[i].s.duration);
        }
```

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

}
/*Design a system to store and manage weld type configurations using
structures for weld type details, unions for variable parameters (e.g.,
voltage or current), and arrays for multiple configurations.
```

```c
/*Specifications:
Structure: Stores weld type ID, name, voltage, and current.
Union: Represents either voltage or current as a variable parameter.
Array: Holds multiple weld type configurations.
const Pointers: Protect weld type details.
Double Pointers: Manage dynamic allocation of weld configurations.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
 float voltage;
    float current;
};
struct Weld
{
    char id[10];
    char name[10];
    int type;//0.voltage 1.current;
    union Parameter s;

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

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

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

```
            printf("Current : %.2f\n",c[i].s.current);
        }

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

}
```

```c
/*Develop a program to manage settings for welding machines, including
mode selection, input voltage range, and speed adjustments.
Specifications:
Structure: Contains machine ID, mode, speed, and input voltage range.
Array: Stores settings for multiple machines.
Strings: Represent machine modes.
const Pointers: Prevent modifications to critical machine settings.
Double Pointers: Allocate and manage machine setting records
dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Machine
{
    char id[10];
    char mode[10];
    float speed;
    float voltage;

};
void add(struct Machine **s,int *count)
{
    printf("Enter Machine id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter mode name :");
    scanf("%s",(*s)[*count].mode);
    printf("Enter speed : ");
    scanf("%f",&(*s)[*count].speed);
    printf("Enter voltage : ");
    scanf("%f",&(*s)[*count].voltage);
(*count)++;
}
void delete(struct Machine **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Machine id : %s | Mode : %s | Speed : %.2f | Voltage :
%.2f \n",s[i].id,s[i].mode,s[i].speed,s[i].voltage);

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

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

}
/*Create a system to track ongoing welding processes using structures for
process metadata, unions for variable process metrics
 (e.g., heat input or arc length), and arrays for process data storage.
Specifications:
Structure: Stores process ID, material, and welder name.
Union: Represents either heat input or arc length.
Array: Stores process data for multiple welding tasks.
const Pointers: Protect metadata for ongoing processes.
Double Pointers: Manage dynamic process records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float heat;
    float arc;;
};
struct Process
{
```

```
    char id[10];
    char name[10];
    char material[10];
    int type;//0.heat 1.arc;
    union Parameter s;

};
void add(struct Process **c,int *count)
{
    printf("Enter process id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material name :");
    scanf("%s",(*c)[*count].material);
    printf("Enter welder name :");
    scanf("%s",(*c)[*count].name);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter heat input : ");
    scanf("%f",&(*c)[*count].s.heat);
    }
    else
    {
        printf("Enter arc length : ");
       scanf("%f",&(*c)[*count].s.arc);
    }
(*count)++;
}
void delete(struct Process **c,int *count,const char *id)
{
    int indefficiency=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            indefficiency=i;
            break;
        }
    }
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Process id : %s | Material: %s | Welder name : %s |
",c[i].id,c[i].material,c[i].name);
        if(c[i].type==0)
        {
            printf("Heat input : %.2f\n",c[i].s.heat);
        }
        else
        {
            printf("Arc length : %.2f\n",c[i].s.arc);
        }

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

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

}
/*Design a program to analyze weld bead geometry using structures for
geometry details, arrays for measurements,
and unions for different parameters like width, depth, and height.
Specifications:
Structure: Contains bead ID, material, and geometry type.
Union: Represents bead width, depth, or height.
Array: Stores geometry measurements.
const Pointers: Protect geometry data.
Double Pointers: Allocate and manage bead records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
```

```
    float depth;
    float height;
};
struct Beead
{
    char id[10];
    char material[10];
    float width;
    int type;//0.depth 1.height;
    union Parameter s;

};
void add(struct Beead **c,int *count)
{
    printf("Enter Beead id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material name :");
    scanf("%s",(*c)[*count].material);
    printf("Enter width : ");
    scanf("%f",&(*c)[*count].width);
    printf("Enter type : ");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter depth : ");
    scanf("%f",&(*c)[*count].s.depth);
    }
    else
    {
        printf("Enter height : ");
       scanf("%f",&(*c)[*count].s.height);
    }
(*count)++;
}
void delete(struct Beead **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
```

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

printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Bead id : %s | Material: %s | Width : %.2f |
",c[i].id,c[i].material,c[i].width);
        if(c[i].type==0)
        {
            printf("Depth : %.2f\n",c[i].s.depth);
        }
        else
        {
            printf("Height : %.2f\n",c[i].s.height);
        }

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

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

}
/*Develop a system to manage inventory for welding consumables, including
electrodes, filler materials, and fluxes.
Specifications:
Structure: Stores consumable ID, type, and quantity.
Array: Inventory for different consumables.
Strings: Represent consumable types.
const Pointers: Prevent modifications to consumable details.
Double Pointers: Manage inventory records dynamically.*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
struct Consumable
{
    char id[10];
    char type[10];
    int quantity;

};
void add(struct Consumable **s,int *count)
{
    printf("Enter Consumable id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter consumable type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter  quantity : ");
    scanf("%d",&(*s)[*count].quantity);

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

```
(*count)--;
}
void display(struct Consumable *s,int count)
{
    printf("\nConsumable details \n");

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Consumable id : %s | Type : %s | Quantity :
%d\n",s[i].id,s[i].type,s[i].quantity);

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

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

}
/*Create a program to track safety equipment for welding personnel using
structures for equipment details, arrays for availability status,
and strings for equipment names.
Specifications:
Structure: Holds equipment ID, type, and usage frequency.
Array: Availability status for multiple equipment items.
Strings: Equipment names.
const Pointers: Protect safety equipment data.
Double Pointers: Allocate dynamic safety equipment records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Equipment
{
    char id[10];
    char type[10];
    char name[10];
    int frequency;

};
void add(struct Equipment **s,int *count)
{
    printf("Enter Equipment id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter name :");
    scanf("%s",(*s)[*count].name);
    printf("Enter Equipment type :");
    scanf("%s",(*s)[*count].type);
    printf("Enter  frequency : ");
    scanf("%d",&(*s)[*count].frequency);
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Equipment id : %s | Equipment name : %s |Type : %s |
frequency : %d\n",s[i].id,s[i].name,s[i].type,s[i].frequency);

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

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

}
/*Design a system to classify welding defects using structures for defect
data, arrays for sample analysis, and unions for
 defect types like porosity, cracking, or spatter.
Specifications:
Structure: Stores defect ID, type, and severity level.
Union: Represents defect types.
Array: Sample analysis data.
```

```c
const Pointers: Protect defect classifications.
Double Pointers: Manage defect data dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float porosity;
    int cracks;
};
struct Defect
{
    char id[10];
    int severity;
    int type;//0.porosity 1.crack;
    union Parameter s;

};
void add(struct Defect **c,int *count)
{
    printf("Enter Defect id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter severity level :");
    scanf("%d",&(*c)[*count].severity);
    printf("Enter defect type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter porosity : ");
    scanf("%f",&(*c)[*count].s.porosity);
    }
    else
    {
        printf("Enter no:cracks : ");
       scanf("%d",&(*c)[*count].s.cracks);
    }
(*count)++;
}
void delete(struct Defect **c,int *count,const char *id)
{
```

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

printf("-------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Defect id : %s | Severity : %d | ",c[i].id,c[i].severity);
        if(c[i].type==0)
        {
            printf("Porosity : %.2f\n",c[i].s.porosity);
        }
        else
        {
            printf("Cracks : %d\n",c[i].s.cracks);
        }

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

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

}
/*Develop a program to analyze the performance of arc welding processes
using structures for performance metrics, arrays for output data,
and unions for variable factors like arc stability and penetration depth.
Specifications:
Structure: Contains performance ID, material type, and current setting.
Union: Represents arc stability or penetration depth.
```

```c
/*Array: Output data.
const Pointers: Protect performance configurations.
Double Pointers: Manage dynamic performance data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float stability;
    float depth;
};
struct Performance
{
    char id[10];
    char material[10];
    char setting[10];
    int type;//0.stability 1.depth;
    union Parameter s;

};
void add(struct Performance **c,int *count)
{
    printf("Enter Performance id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material  : ");
    scanf("%s",(*c)[*count].material);
    printf("Enter setting : ");
    scanf("%s",(*c)[*count].setting);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter stability : ");
    scanf("%f",&(*c)[*count].s.stability);
    }
    else
    {
        printf("Enter depth : ");
       scanf("%f",&(*c)[*count].s.depth);
    }
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Performance id : %s | Material : %s | Setting : %s
",c[i].id,c[i].material,c[i].setting);
        if(c[i].type==0)
        {
            printf("Stability : %.2f\n",c[i].s.stability);
        }
        else
        {
```

```
            printf("Depth : %.2f\n",c[i].s.depth);
        }

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

}
```

```c
/*Create a program to optimize welding schedules using structures for task
details, arrays for time slots, and strings for task names.
Specifications:
Structure: Holds task ID, priority, and duration.
Array: Time slots for scheduling.
Strings: Task names.
const Pointers: Protect task details.
Double Pointers: Allocate and manage task records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Schedule
{
    char id[10];
    char due[20];
    char name[10];
    int priority;

};
void add(struct Schedule **s,int *count)
{
    printf("Enter Schedule id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter name :");
    scanf("%s",(*s)[*count].name);
    printf("Enter Schedule duration :");
    getchar();
    fgets((*s)[*count].due,20,stdin);
    printf("Enter  priority : ");
    scanf("%d",&(*s)[*count].priority);
    (*count)++;
}
void delete(struct Schedule **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Schedule id : %s | Task name : %s |Duration : %s |
priority : %d\n",s[i].id,s[i].name,s[i].due,s[i].priority);

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

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

}
/*Develop a system to automate the inspection of welds using structures
for inspection details, arrays for measurement data,
and unions for different defect parameters.
Specifications:
Structure: Stores inspection ID, method, and results.
Union: Represents defect parameters like size or location.
Array: Measurement data.
const Pointers: Protect inspection configurations.
Double Pointers: Manage inspection records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float size;
    char location[10];
};
struct Inspection
```

```
{
    char id[10];
    char method[10];
    int type;//0.size 1.location;
    union Parameter s;

};
void add(struct Inspection **c,int *count)
{
    printf("Enter Inspection id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter method  : ");
    scanf("%s",(*c)[*count].method);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter size : ");
    scanf("%f",&(*c)[*count].s.size);
    }
    else
    {
        printf("Enter location : ");
       scanf("%s",(*c)[*count].s.location);
    }
(*count)++;
}
void delete(struct Inspection **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Inspection id : %s | Method : %s |",c[i].id,c[i].method);
        if(c[i].type==0)
        {
            printf("Size : %.2f\n",c[i].s.size);
        }
        else
        {
            printf("Location : %s\n",c[i].s.location);
        }

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

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

}
/*Design a control system for welding robots using structures for robot
configurations, arrays for motion data, and strings for robot types.
Specifications:
Structure: Holds robot ID, configuration, and status.
Array: Motion data for robotic operations.
Strings: Robot types.
const Pointers: Protect robot configurations.
Double Pointers: Allocate and manage robot records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Robot
{
    char id[10];
    char configuration[20];
    char status[10];
```

```
};
void add(struct Robot **s,int *count)
{
    printf("Enter Robot id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter status :");
    scanf("%s",(*s)[*count].status);
    printf("Enter Robot configuration :");
    getchar();
    fgets((*s)[*count].configuration,20,stdin);
    (*count)++;
}
void delete(struct Robot **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Robot not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*s)[i]=(*s)[i+1];
    }
(*count)--;
}
void display(struct Robot *s,int count)
{
    printf("\nRobot details \n");
```

```
printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Robot id : %s | Robot status : %s | Configuration :
%s\n",s[i].id,s[i].status,s[i].configuration);

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

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

}
/*Create a data logger for weld quality metrics using structures for weld
details, arrays for quality data, and unions for different quality
parameters.
Specifications:
Structure: Stores weld ID, material, and quality score.
Union: Represents different quality parameters.
Array: Quality data for multiple welds.
const Pointers: Protect weld details.
Double Pointers: Manage dynamic quality data.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float size;
    float crack;
};
struct Quality
{
    char id[10];
    char material[10];
    int score;
    int type;//0.size 1.crack;
    union Parameter s;

};
void add(struct Quality **c,int *count)
{
    printf("Enter Quality id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter material  : ");
    scanf("%s",(*c)[*count].material);
    printf("Enter score : ");
    scanf("%d",&(*c)[*count].score);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
```

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

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

```
    for(int i=0;i<count;i++)
    {
        printf("Quality id : %s | material : %s |",c[i].id,c[i].material);
        if(c[i].type==0)
        {
            printf("Size : %.2f\n",c[i].s.size);
        }
        else
        {
            printf("Crack : %.2f\n",c[i].s.crack);
        }

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

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

}
/*Develop a program to analyze thermal input in welding using structures
for thermal details, arrays for time-temperature data,
and unions for heat input variables.
Specifications:
Structure: Holds thermal input ID, current, and voltage.
Union: Represents heat input or time-temperature correlation.
Array: Time-temperature data.
const Pointers: Protect thermal input data.
Double Pointers: Manage thermal data dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float ti;
    float tt;
};
struct Thermal
{
    char id[10];
    int v;
    int c;
    int type;//0.ti 1.tt;
    union Parameter s;

};
void add(struct Thermal **c,int *count)
{
    printf("Enter Thermal id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter voltage  : ");
    scanf("%d",&(*c)[*count].v);
```

```
    printf("Enter current : ");
    scanf("%d",&(*c)[*count].c);
    printf("Enter type :");
    scanf("%d",&(*c)[*count].type);
    if((*c)[*count].type==0)
    {
        printf("Enter thermal input : ");
    scanf("%f",&(*c)[*count].s.ti);
    }
    else
    {
        printf("Enter time -temperature corelation : ");
       scanf("%f",&(*c)[*count].s.tt);
    }
(*count)++;
}
void delete(struct Thermal **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*c)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Thermal data not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*c)[i]=(*c)[i+1];
    }
(*count)--;
}
void display(struct Thermal *c,int count)
{
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Thermal id : %s | Voltage : %d | Current : %d |
",c[i].id,c[i].v,c[i].c);
        if(c[i].type==0)
        {
            printf("Thermal input : %.2f\n",c[i].s.ti);
        }
        else
        {
            printf("Time temperature : %.2f\n",c[i].s.tt);
        }

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

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

}

/*Create a program to manage welding procedure specifications using
structures for procedure details, arrays for parameters, and strings for
procedure types.
Specifications:
Structure: Contains procedure ID, material, and joint type.
Array: Welding parameters.
Strings: Procedure types.
const Pointers: Protect procedure details.
Double Pointers: Allocate dynamic procedure records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Procedure
{
    char id[10];
    char material[20];
    char type[10];
};
void add(struct Procedure **s,int *count)
{
    printf("Enter Procedure id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter material :");
    scanf("%s",(*s)[*count].material);
    printf("Enter joint type :");
    scanf("%s",(*s)[*count].type);
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Procedure id : %s | Material : %s | Joint type : %s
\n",s[i].id,s[i].material,s[i].type);

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

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

}
/*Design a tracker for joint designs in welding using structures for joint
details, arrays for dimensions, and unions for variable joint parameters.
Specificaangleons:
Structure: Stores joint ID, mtype, and angle.
Union: Represents joint parameters.
Array: Dimensions for mulangleple joints.
const Pointers: Protect joint data.
Double Pointers: Manage joint records dynamically.*/
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
union Parameter
{
    float angle;
    float length;
};
struct Joint
{
    char id[10];
    char type[10];
    int mtype;//0.angle 1.length;
    union Parameter s;

};
void add(struct Joint **c,int *count)
{
    printf("Enter Joint id : ");
    scanf("%s",(*c)[*count].id);
    printf("Enter joint type : ");
    scanf("%s",(*c)[*count].type);
    printf("Enter mtype :");
    scanf("%d",&(*c)[*count].mtype);
    if((*c)[*count].mtype==0)
    {
        printf("Enter angle : ");
    scanf("%f",&(*c)[*count].s.angle);
    }
    else
    {
        printf("Enter length : ");
       scanf("%f",&(*c)[*count].s.length);
    }
(*count)++;
}
void delete(struct Joint **c,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Joint id : %s | Joint type : %s ",c[i].id,c[i].type);
        if(c[i].mtype==0)
        {
            printf("Joint input : %.2f\n",c[i].s.angle);
        }
        else
        {
            printf("Joint angle : %.2f\n",c[i].s.length);
        }

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

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

}

/*Develop a program to select filler metals using structures for metal
properties, arrays for test results, and strings for metal names.
Specifications:
Structure: Holds filler metal ID, composition, and diameter.
Array: Test results for filler metals.
Strings: Filler metal names.
const Pointers: Protect filler metal data.
```

```c
/*Double Pointers: Allocate and manage filler metal records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Metal
{
    char id[10];
    char name[10];
    char composition[20];
    float diameter;

};
void add(struct Metal **s,int *count)
{
    printf("Enter Metal id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter name : ");
    scanf("%s",(*s)[*count].name);
    printf("Enter diameter :");
    scanf("%f",&(*s)[*count].diameter);
    printf("Enter Metal composition :");
    getchar();
    fgets((*s)[*count].composition,20,stdin);
    (*count)++;
}
void delete(struct Metal **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Metal not found\n");
        return;
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Metal id : %s | Metal name : %s | composition : %s |
Diameter : %.2f \n",s[i].id,s[i].name,s[i].composition,s[i].diameter);

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

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

}
/*Create a system to configure welding power sources using structures for
source details, arrays for power settings, and strings for source types.
Specifications:
Structure: Contains source ID, type, and capacity.
Array: Power settings for multiple sources.
Strings: Source types.
const Pointers: Protect power source configurations.
Double Pointers: Allocate and manage source records.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Source
{
    char id[10];
    char type[10];
    float capacity;

};
void add(struct Source **s,int *count)
{
    printf("Enter Source id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter type : ");
    scanf("%s",(*s)[*count].type);
    printf("Enter capacity :");
    scanf("%f",&(*s)[*count].capacity);
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Source id : %s | Source type : %s | Capacity : %.2f
\n",s[i].id,s[i].type,s[i].capacity);

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

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

}
/*Develop a program to assess the skills of welders using structures for
skill data, arrays for test results, and strings for skill levels.
Specifications:
Structure: Holds welder ID, name, and skill score.
Array: Test results for skill assessment.
Strings: Skill levels.
const Pointers: Protect skill assessment data.
Double Pointers: Manage skill records dynamically.*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
struct Welder
{
    char id[10];
    char name[10];
    float score;

};
void add(struct Welder **s,int *count)
{
    printf("Enter Welder id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter name : ");
    scanf("%s",(*s)[*count].name);
    printf("Enter score :");
    scanf("%f",&(*s)[*count].score);
    (*count)++;
}
void delete(struct Welder **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Welder not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*s)[i]=(*s)[i+1];
    }
(*count)--;
```

```
}
void display(struct Welder *s,int count)
{
    printf("\n Welderdetails \n");

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Welder id : %s | Welder name : %s | Score : %.2f
\n",s[i].id,s[i].name,s[i].score);

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

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

}
/*Design a program to analyze welding arc stability using structures for
stability data, arrays for voltage readings, and unions for different
stability metrics.
Specifications:
Structure: Contains stability ID, voltage, and current.
Union: Represents stability metrics like arc length or consistency.
Array: Voltage readings.
const Pointers: Protect stability data.
Double Pointers: Allocate and manage stability records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Stability
{
    char id[10];
    float current;
    float voltage;

};
void add(struct Stability **s,int *count)
{
    printf("Enter Stability id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter current : ");
    scanf("%f",&(*s)[*count].current);
    printf("Enter voltage :");
    scanf("%f",&(*s)[*count].voltage);
    (*count)++;
}
void delete(struct Stability **s,int *count,const char *id)
{
    int index=-1;
```

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

printf("------------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Stability id : %s |  Current : %.2f | Voltage : %.2f
\n",s[i].id,s[i].current,s[i].voltage);

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

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

}
/*Create a simulation system for welding training using structures for
training details, arrays for progress data, and strings for training
modules.
Specifications:
Structure: Stores training ID, module name, and trainee progress.
Array: Progress data for multiple trainees.
Strings: Training module names.
const Pointers: Protect training details.
Double Pointers: Manage training records dynamically.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Training
```

```
{
    char id[10];
    char module[10];
    char progress[10];

};
void add(struct Training **s,int *count)
{
    printf("Enter Training id : ");
    scanf("%s",(*s)[*count].id);
    printf("Enter module : ");
    scanf("%s",(*s)[*count].module);
    printf("Enter progress :");
    scanf("%s",(*s)[*count].progress);
    (*count)++;
}
void delete(struct Training **s,int *count,const char *id)
{
    int index=-1;
    for(int i=0;i<*count;i++)
    {
        if(strcmp((*s)[i].id,id)==0)
        {
            index=i;
            break;
        }
    }
    if(index==-1)
    {
        printf("Training not found\n");
        return;
    }
    for(int i=index;i<*count;i++)
    {
        (*s)[i]=(*s)[i+1];
    }
(*count)--;
}
void display(struct Training *s,int count)
{
```

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

printf("------------------------------------------------------------------
----------------------\n");
    for(int i=0;i<count;i++)
    {
        printf("Training id : %s |  module : %s | progress : %s
\n",s[i].id,s[i].module,s[i].progress);

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

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

}
```