# 第11章 構造体とポインタ - 演習問題

# 演習問題

## 基礎問題

### 演習11-1: 学生情報管理

学生の情報（ID、名前、年齢、成績）を格納する構造体を定義し、ポインタを使って情報を表示・更新するプログラムを作成してください。

**要件:** - 学生情報を表す構造体を定義 - 学生情報を入力する関数を作成 - 学生情報を表示する関数を作成（ポインタ引数） - 成績を更新する関数を作成（ポインタ引数）

**期待される動作例:**

学生情報を入力してください:  
ID: 1001  
名前: 田中太郎  
年齢: 20  
成績: 85.5  
  
=== 学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 85.5点  
  
新しい成績を入力してください: 90.0  
成績を更新しました。  
  
=== 更新後の学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点

### 演習11-2: 座標計算

2D座標を表す構造体を定義し、2点間の距離を計算する関数をポインタを使って実装してください。

**要件:** - 座標を表す構造体（x, y）を定義 - 2点間の距離を計算する関数を作成（ポインタ引数） - 座標を移動する関数を作成（ポインタで更新） - math.hのsqrt関数を使用

**期待される動作例:**

点1の座標を入力 (x y): 0 0  
点2の座標を入力 (x y): 3 4  
  
点1: (0, 0)  
点2: (3, 4)  
2点間の距離: 5.00  
  
点1を移動します。  
移動量を入力 (dx dy): 1 1  
  
移動後の点1: (1, 1)  
新しい距離: 3.61

### 演習11-3: 商品管理

商品情報（コード、名前、価格、在庫）の構造体を作成し、構造体配列で複数商品を管理するプログラムを作成してください。

**要件:** - 商品情報を表す構造体を定義 - 最大10個の商品を管理できる配列を用意 - 商品を追加する関数を作成 - 全商品を表示する関数を作成 - 在庫を更新する関数を作成（商品コードで検索）

**期待される動作例:**

=== 商品管理システム ===  
1. 商品追加  
2. 商品一覧表示  
3. 在庫更新  
0. 終了  
選択: 1  
  
商品コード: 101  
商品名: ノートPC  
価格: 98000  
在庫数: 5  
商品を追加しました。  
  
選択: 2  
=== 商品一覧 ===  
コード: 101  
商品名: ノートPC  
価格: 98000円  
在庫: 5個

## 応用問題

### 演習11-4: 従業員データベース

従業員情報と部署情報をネストした構造体で管理し、部署別の給与統計を算出するプログラムを作成してください。

**要件:** - 部署情報を表す構造体（部署名、部署コード） - 従業員情報を表す構造体（ID、名前、部署情報、給与） - 部署別の平均給与を計算する関数 - 最高給与の従業員を検索する関数 - 構造体ポインタ配列を使用

### 演習11-5: 図書管理システム

本の情報（タイトル、著者、出版年、貸出状況）を管理し、検索・貸出・返却機能を実装してください。

**要件:** - 書籍情報を表す構造体を定義 - 貸出状況を管理（貸出中フラグ、借りた人のID） - タイトルで検索する関数 - 貸出処理を行う関数 - 返却処理を行う関数 - 貸出中の本一覧を表示する関数

### 演習11-6: 成績管理システム

学生と科目の構造体を使って、学生別・科目別の成績統計を管理するプログラムを作成してください。

**要件:** - 科目情報を表す構造体（科目名、科目コード、単位数） - 成績情報を表す構造体（学生ID、科目コード、点数、評価） - 学生別の平均点を計算する関数 - 科目別の平均点を計算する関数 - GPA計算機能（A=4.0, B=3.0, C=2.0, D=1.0, F=0.0）

## 発展問題

### 演習11-7: 動的配列システム

構造体ポインタ配列を動的に拡張できるシステムを実装してください。

**要件:** - 初期容量を持つ構造体ポインタ配列 - 容量が不足したら自動的に拡張する機能 - メモリ管理を適切に行う - 要素の追加・削除機能

### 演習11-8: データソート

構造体ポインタ配列を複数の条件（名前、年齢、成績など）でソートできるプログラムを作成してください。

**要件:** - 複数のソート基準を選択可能 - 昇順・降順の切り替え機能 - ソート用の比較関数を複数実装 - qsort関数を使用

### 演習11-9: 階層データ構造

会社組織（部署→チーム→従業員）のような階層構造を構造体とポインタで表現してください。

**要件:** - 部署、チーム、従業員の構造体を定義 - 階層関係をポインタで表現 - 組織図を表示する関数 - 特定の部署・チームの人数を集計する関数

## 提出方法

1. 各問題に対して別々のCファイルを作成
2. ファイル名は ex11\_1\_student.c, ex11\_2\_coordinate.c のように命名
3. 各プログラムの冒頭にコメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認

## コンパイル例

gcc -std=c90 -Wall -Wextra -pedantic ex11\_1\_student.c -o ex11\_1\_student  
./ex11\_1\_student

## ヒント

* 構造体のサイズに注意（パディング）
* ポインタ経由でのメンバーアクセスには->演算子を使用
* 構造体の比較は直接できないため、メンバーごとに比較
* 文字列のコピーにはstrcpy関数を使用
* 構造体を関数に渡す際は、効率のためポインタを使用することを推奨

# 解答例

このディレクトリには演習問題の解答例が含まれています。

## 解答ファイル一覧

### 基礎問題

1. **演習11-1: 学生情報管理**
   * <ex11_1_student.c> - C90準拠版
   * <ex11_1_student_c99.c> - C99準拠版
   * 構造体とポインタを使った基本的なデータ管理
2. **演習11-2: 座標計算**
   * <ex11_2_coordinate.c> - C90準拠版
   * <ex11_2_coordinate_c99.c> - C99準拠版
   * 構造体で座標を表現し、距離計算を実装
3. **演習11-3: 商品管理**
   * <ex11_3_product.c> - C90準拠版
   * <ex11_3_product_c99.c> - C99準拠版
   * 構造体配列による商品在庫管理システム

### 応用問題

1. **演習11-4: 従業員データベース**
   * ネストした構造体と統計計算
   * （未実装）
2. **演習11-5: 図書管理システム**
   * 検索・貸出・返却機能の実装
   * （未実装）
3. **演習11-6: 成績管理システム**
   * 学生別・科目別の成績統計とGPA計算
   * （未実装）

### 発展問題

1. **演習11-7: 動的配列システム**
   * 動的メモリ管理による拡張可能な配列
   * （未実装）
2. **演習11-8: データソート**
   * 複数条件でのソート機能
   * （未実装）
3. **演習11-9: 階層データ構造**
   * 会社組織の階層構造表現
   * （未実装）

## 学習のポイント

### C90版とC99版の主な違い

1. **構造体の初期化**
   * C90: 順序通りの初期化のみ
   * C99: 指定イニシャライザが使用可能
2. **bool型**
   * C90: intで代用
   * C99: stdbool.hのbool型
3. **変数宣言**
   * C90: ブロックの先頭でのみ
   * C99: 使用箇所で宣言可能
4. **可変長配列（VLA）**
   * C90: 未対応
   * C99: 実行時にサイズ決定可能
5. **inline関数**
   * C90: 未対応
   * C99: inline修飾子で最適化

### 各演習の重要概念

#### 演習11-1〜11-3（基礎）

* 構造体の定義と初期化
* ドット演算子とアロー演算子
* 構造体配列の操作
* ポインタによる効率的なデータ渡し

#### 演習11-4〜11-6（応用）

* ネストした構造体
* 構造体ポインタ配列
* 複雑なデータ構造の管理
* 統計処理とデータ分析

#### 演習11-7〜11-9（発展）

* 動的メモリ管理
* qsortによる汎用ソート
* 階層的なデータ構造
* より高度なポインタ操作

## コンパイルと実行

# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic ex11\_1\_student.c -o ex11\_1\_student  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic ex11\_1\_student\_c99.c -o ex11\_1\_student\_c99  
  
# 座標計算（数学ライブラリをリンク）  
gcc -std=c90 -Wall -Wextra -pedantic ex11\_2\_coordinate.c -o ex11\_2\_coordinate -lm  
  
# 実行  
./ex11\_1\_student

## 注意事項

* 構造体のサイズはパディングの影響を受けることがある
* 構造体の直接比較はできない（メンバーごとに比較）
* 文字列メンバーへの代入にはstrcpyを使用
* 大きな構造体は値渡しより、ポインタ渡しが効率的
* 動的メモリを使用する場合は、必ず解放処理を実装 ## ex11\_1\_student.c

/\*  
 \* ファイル名: ex11\_1\_student.c  
 \* 演習11-1: 学生情報管理  
 \* 説明: 構造体とポインタを使った学生情報の管理  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_NAME\_LEN 50  
  
/\* 学生情報を表す構造体 \*/  
struct Student {  
 int id;  
 char name[MAX\_NAME\_LEN];  
 int age;  
 double score;  
};  
  
/\* 学生情報を入力する関数（デモ版） \*/  
void input\_student(struct Student \*student)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("学生情報を入力してください:\n");  
   
 /\* デモ用: 固定値を設定 \*/  
 student->id = 1001;  
 student->age = 20;  
 student->score = 85.5;  
 strcpy(student->name, "田中太郎");  
   
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d\n", student->age);  
 printf("成績: %.1f\n", student->score);  
}  
  
/\* 学生情報を表示する関数 \*/  
void display\_student(const struct Student \*student)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("\n=== 学生情報 ===\n");  
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d歳\n", student->age);  
 printf("成績: %.1f点\n", student->score);  
}  
  
/\* 成績を更新する関数 \*/  
void update\_score(struct Student \*student, double new\_score)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 if (new\_score < 0.0 || new\_score > 100.0) {  
 printf("エラー: 成績は0〜100の範囲で入力してください\n");  
 return;  
 }  
   
 double old\_score = student->score;  
 student->score = new\_score;  
   
 printf("\n成績を更新しました。\n");  
 printf("変更前: %.1f点 → 変更後: %.1f点\n", old\_score, new\_score);  
}  
  
/\* 成績の評価を返す関数 \*/  
char get\_grade(double score)  
{  
 if (score >= 90.0) return 'A';  
 if (score >= 80.0) return 'B';  
 if (score >= 70.0) return 'C';  
 if (score >= 60.0) return 'D';  
 return 'F';  
}  
  
/\* 詳細情報を表示する関数 \*/  
void display\_detailed\_info(const struct Student \*student)  
{  
 if (student == NULL) {  
 return;  
 }  
   
 printf("\n=== 詳細な学生情報 ===\n");  
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d歳\n", student->age);  
 printf("成績: %.1f点 (評価: %c)\n", student->score, get\_grade(student->score));  
   
 /\* 成績に応じたコメント \*/  
 if (student->score >= 90.0) {  
 printf("コメント: 大変優秀です！\n");  
 } else if (student->score >= 80.0) {  
 printf("コメント: 良い成績です。\n");  
 } else if (student->score >= 70.0) {  
 printf("コメント: まずまずの成績です。\n");  
 } else if (student->score >= 60.0) {  
 printf("コメント: もう少し頑張りましょう。\n");  
 } else {  
 printf("コメント: 努力が必要です。\n");  
 }  
}  
  
int main(void)  
{  
 struct Student student;  
 double new\_score = 90.0; /\* デモ用: 固定値 \*/  
   
 printf("===== 学生情報管理システム =====\n\n");  
   
 /\* 学生情報の入力 \*/  
 input\_student(&student);  
   
 /\* 学生情報の表示 \*/  
 display\_student(&student);  
   
 /\* 成績更新のデモ \*/  
 printf("\n成績を更新しますか？ (y/n): y\n");  
 printf("新しい成績を入力してください: %.1f\n", new\_score);  
   
 /\* 成績の更新 \*/  
 update\_score(&student, new\_score);  
   
 /\* 更新後の情報表示 \*/  
 printf("\n=== 更新後の学生情報 ===\n");  
 printf("ID: %d\n", student.id);  
 printf("名前: %s\n", student.name);  
 printf("年齢: %d歳\n", student.age);  
 printf("成績: %.1f点\n", student.score);  
   
 /\* 詳細情報の表示 \*/  
 display\_detailed\_info(&student);  
   
 /\* 追加機能：複数の学生の比較 \*/  
 printf("\n\n=== 複数学生の管理デモ ===\n");  
   
 struct Student students[3] = {  
 {2001, "田中太郎", 20, 85.5},  
 {2002, "佐藤花子", 21, 92.0},  
 {2003, "山田次郎", 19, 78.0}  
 };  
   
 int i;  
 double total\_score = 0.0;  
 struct Student \*top\_student = &students[0];  
   
 printf("\n全学生の一覧:\n");  
 for (i = 0; i < 3; i++) {  
 printf("%d. %s (%.1f点)\n",   
 students[i].id, students[i].name, students[i].score);  
 total\_score += students[i].score;  
   
 /\* 最高得点の学生を記録 \*/  
 if (students[i].score > top\_student->score) {  
 top\_student = &students[i];  
 }  
 }  
   
 printf("\n統計情報:\n");  
 printf("平均点: %.1f点\n", total\_score / 3);  
 printf("最高得点: %s (%.1f点)\n", top\_student->name, top\_student->score);  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 学生情報管理システム =====  
  
学生情報を入力してください:  
ID: 1001  
名前: 田中太郎  
年齢: 20  
成績: 85.5  
  
=== 学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 85.5点  
  
成績を更新しますか？ (y/n): y  
新しい成績を入力してください: 90.0  
  
成績を更新しました。  
変更前: 85.5点 → 変更後: 90.0点  
  
=== 更新後の学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点  
  
=== 詳細な学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点 (評価: A)  
コメント: 大変優秀です！  
  
=== 複数学生の管理デモ ===  
  
全学生の一覧:  
2001. 田中太郎 (85.5点)  
2002. 佐藤花子 (92.0点)  
2003. 山田次郎 (78.0点)  
  
統計情報:  
平均点: 85.2点  
最高得点: 佐藤花子 (92.0点)  
\*/

## ex11\_2\_coordinate.c

/\*  
 \* ファイル名: ex11\_2\_coordinate.c  
 \* 演習11-2: 座標計算  
 \* 説明: 2D座標を表す構造体を定義し、2点間の距離を計算  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <math.h>  
  
/\* 2D座標を表す構造体 \*/  
struct Point2D {  
 double x;  
 double y;  
};  
  
/\* 座標を表示する関数 \*/  
void display\_point(const struct Point2D \*point, const char \*name)  
{  
 if (point == NULL || name == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("%s: (%.2f, %.2f)\n", name, point->x, point->y);  
}  
  
/\* 2点間の距離を計算する関数（ポインタ引数） \*/  
double calculate\_distance(const struct Point2D \*p1, const struct Point2D \*p2)  
{  
 double dx, dy;  
   
 if (p1 == NULL || p2 == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return -1.0;  
 }  
   
 dx = p2->x - p1->x;  
 dy = p2->y - p1->y;  
   
 return sqrt(dx \* dx + dy \* dy);  
}  
  
/\* 座標を移動する関数（ポインタで更新） \*/  
void move\_point(struct Point2D \*point, double dx, double dy)  
{  
 if (point == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 point->x += dx;  
 point->y += dy;  
}  
  
/\* 中点を計算する関数 \*/  
struct Point2D calculate\_midpoint(const struct Point2D \*p1, const struct Point2D \*p2)  
{  
 struct Point2D midpoint;  
   
 if (p1 == NULL || p2 == NULL) {  
 midpoint.x = 0.0;  
 midpoint.y = 0.0;  
 return midpoint;  
 }  
   
 midpoint.x = (p1->x + p2->x) / 2.0;  
 midpoint.y = (p1->y + p2->y) / 2.0;  
   
 return midpoint;  
}  
  
/\* 原点からの距離を計算する関数 \*/  
double distance\_from\_origin(const struct Point2D \*point)  
{  
 struct Point2D origin = {0.0, 0.0};  
 return calculate\_distance(&origin, point);  
}  
  
/\* 座標の象限を判定する関数 \*/  
int get\_quadrant(const struct Point2D \*point)  
{  
 if (point == NULL) {  
 return 0;  
 }  
   
 if (point->x > 0 && point->y > 0) {  
 return 1; /\* 第1象限 \*/  
 } else if (point->x < 0 && point->y > 0) {  
 return 2; /\* 第2象限 \*/  
 } else if (point->x < 0 && point->y < 0) {  
 return 3; /\* 第3象限 \*/  
 } else if (point->x > 0 && point->y < 0) {  
 return 4; /\* 第4象限 \*/  
 } else {  
 return 0; /\* 軸上または原点 \*/  
 }  
}  
  
int main(void)  
{  
 struct Point2D point1, point2;  
 struct Point2D midpoint;  
 double distance;  
 double dx, dy;  
 int quadrant;  
   
 printf("===== 2D座標計算プログラム =====\n\n");  
   
 /\* デモ用: 固定値で初期化 \*/  
 printf("点1の座標を入力 (x y): 0 0\n");  
 point1.x = 0.0;  
 point1.y = 0.0;  
   
 printf("点2の座標を入力 (x y): 3 4\n");  
 point2.x = 3.0;  
 point2.y = 4.0;  
   
 /\* 座標の表示 \*/  
 printf("\n");  
 display\_point(&point1, "点1");  
 display\_point(&point2, "点2");  
   
 /\* 2点間の距離を計算 \*/  
 distance = calculate\_distance(&point1, &point2);  
 printf("2点間の距離: %.2f\n", distance);  
   
 /\* 中点の計算 \*/  
 midpoint = calculate\_midpoint(&point1, &point2);  
 display\_point(&midpoint, "\n2点の中点");  
   
 /\* 原点からの距離 \*/  
 printf("\n原点からの距離:\n");  
 printf("点1: %.2f\n", distance\_from\_origin(&point1));  
 printf("点2: %.2f\n", distance\_from\_origin(&point2));  
   
 /\* 座標の移動 \*/  
 printf("\n点1を移動します。\n");  
 printf("移動量を入力 (dx dy): 1 1\n");  
 dx = 1.0;  
 dy = 1.0;  
   
 move\_point(&point1, dx, dy);  
   
 printf("\n移動後の");  
 display\_point(&point1, "点1");  
   
 /\* 新しい距離の計算 \*/  
 distance = calculate\_distance(&point1, &point2);  
 printf("新しい距離: %.2f\n", distance);  
   
 /\* 象限の判定 \*/  
 printf("\n象限の判定:\n");  
 quadrant = get\_quadrant(&point1);  
 printf("点1は");  
 if (quadrant > 0) {  
 printf("第%d象限にあります\n", quadrant);  
 } else {  
 printf("軸上または原点にあります\n");  
 }  
   
 quadrant = get\_quadrant(&point2);  
 printf("点2は");  
 if (quadrant > 0) {  
 printf("第%d象限にあります\n", quadrant);  
 } else {  
 printf("軸上または原点にあります\n");  
 }  
   
 /\* 追加デモ: 複数点の管理 \*/  
 printf("\n\n=== 複数点の管理デモ ===\n");  
   
 struct Point2D points[] = {  
 {1.0, 2.0},  
 {-3.0, 4.0},  
 {-2.0, -5.0},  
 {4.0, -1.0},  
 {0.0, 3.0}  
 };  
 int num\_points = 5;  
 int i, j;  
 double max\_distance = 0.0;  
 int max\_i = 0, max\_j = 1;  
   
 /\* 全ての点を表示 \*/  
 printf("\n登録された点:\n");  
 for (i = 0; i < num\_points; i++) {  
 char name[20];  
 sprintf(name, "点%d", i + 1);  
 display\_point(&points[i], name);  
 }  
   
 /\* 最も離れた2点を見つける \*/  
 for (i = 0; i < num\_points - 1; i++) {  
 for (j = i + 1; j < num\_points; j++) {  
 distance = calculate\_distance(&points[i], &points[j]);  
 if (distance > max\_distance) {  
 max\_distance = distance;  
 max\_i = i;  
 max\_j = j;  
 }  
 }  
 }  
   
 printf("\n最も離れた2点:\n");  
 printf("点%d (%.1f, %.1f) と 点%d (%.1f, %.1f)\n",  
 max\_i + 1, points[max\_i].x, points[max\_i].y,  
 max\_j + 1, points[max\_j].x, points[max\_j].y);  
 printf("距離: %.2f\n", max\_distance);  
   
 /\* 全点の重心を計算 \*/  
 struct Point2D centroid = {0.0, 0.0};  
 for (i = 0; i < num\_points; i++) {  
 centroid.x += points[i].x;  
 centroid.y += points[i].y;  
 }  
 centroid.x /= num\_points;  
 centroid.y /= num\_points;  
   
 printf("\n全点の重心: (%.2f, %.2f)\n", centroid.x, centroid.y);  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 2D座標計算プログラム =====  
  
点1の座標を入力 (x y): 0 0  
点2の座標を入力 (x y): 3 4  
  
点1: (0.00, 0.00)  
点2: (3.00, 4.00)  
2点間の距離: 5.00  
  
2点の中点: (1.50, 2.00)  
  
原点からの距離:  
点1: 0.00  
点2: 5.00  
  
点1を移動します。  
移動量を入力 (dx dy): 1 1  
  
移動後の点1: (1.00, 1.00)  
新しい距離: 3.61  
  
象限の判定:  
点1は第1象限にあります  
点2は第1象限にあります  
  
=== 複数点の管理デモ ===  
  
登録された点:  
点1: (1.00, 2.00)  
点2: (-3.00, 4.00)  
点3: (-2.00, -5.00)  
点4: (4.00, -1.00)  
点5: (0.00, 3.00)  
  
最も離れた2点:  
点2 (-3.0, 4.0) と 点3 (-2.0, -5.0)  
距離: 9.06  
  
全点の重心: (0.00, 0.60)  
\*/

## ex11\_3\_product.c

/\*  
 \* ファイル名: ex11\_3\_product.c  
 \* 演習11-3: 商品管理  
 \* 説明: 商品情報の構造体配列で複数商品を管理  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_PRODUCTS 10  
#define MAX\_NAME\_LEN 50  
  
/\* 商品情報を表す構造体 \*/  
struct Product {  
 int code;  
 char name[MAX\_NAME\_LEN];  
 double price;  
 int stock;  
};  
  
/\* 商品を追加する関数 \*/  
int add\_product(struct Product products[], int \*count,   
 int code, const char \*name, double price, int stock)  
{  
 int i;  
   
 if (\*count >= MAX\_PRODUCTS) {  
 printf("エラー: これ以上商品を追加できません（最大%d個）\n", MAX\_PRODUCTS);  
 return 0;  
 }  
   
 if (name == NULL || strlen(name) == 0) {  
 printf("エラー: 商品名が無効です\n");  
 return 0;  
 }  
   
 if (price < 0) {  
 printf("エラー: 価格は0以上である必要があります\n");  
 return 0;  
 }  
   
 if (stock < 0) {  
 printf("エラー: 在庫数は0以上である必要があります\n");  
 return 0;  
 }  
   
 /\* 商品コードの重複チェック \*/  
 for (i = 0; i < \*count; i++) {  
 if (products[i].code == code) {  
 printf("エラー: 商品コード %d は既に登録されています\n", code);  
 return 0;  
 }  
 }  
   
 /\* 商品を追加 \*/  
 products[\*count].code = code;  
 strncpy(products[\*count].name, name, MAX\_NAME\_LEN - 1);  
 products[\*count].name[MAX\_NAME\_LEN - 1] = '\0';  
 products[\*count].price = price;  
 products[\*count].stock = stock;  
   
 (\*count)++;  
 printf("商品を追加しました。\n");  
   
 return 1;  
}  
  
/\* 全商品を表示する関数 \*/  
void display\_all\_products(const struct Product products[], int count)  
{  
 int i;  
 double total\_value = 0.0;  
   
 if (count == 0) {  
 printf("登録されている商品はありません。\n");  
 return;  
 }  
   
 printf("\n=== 商品一覧 ===\n");  
 printf("%-8s %-20s %10s %8s\n", "コード", "商品名", "価格", "在庫");  
 printf("--------------------------------------------------------\n");  
   
 for (i = 0; i < count; i++) {  
 printf("%-8d %-20s %10.0f円 %8d個\n",  
 products[i].code,  
 products[i].name,  
 products[i].price,  
 products[i].stock);  
 total\_value += products[i].price \* products[i].stock;  
 }  
   
 printf("--------------------------------------------------------\n");  
 printf("登録商品数: %d個\n", count);  
 printf("在庫総額: %.0f円\n", total\_value);  
}  
  
/\* 商品コードで検索する関数 \*/  
struct Product\* find\_product\_by\_code(struct Product products[], int count, int code)  
{  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (products[i].code == code) {  
 return &products[i];  
 }  
 }  
   
 return NULL;  
}  
  
/\* 在庫を更新する関数 \*/  
int update\_stock(struct Product products[], int count, int code, int new\_stock)  
{  
 struct Product \*product;  
 int old\_stock;  
   
 if (new\_stock < 0) {  
 printf("エラー: 在庫数は0以上である必要があります\n");  
 return 0;  
 }  
   
 product = find\_product\_by\_code(products, count, code);  
   
 if (product == NULL) {  
 printf("エラー: 商品コード %d は見つかりませんでした\n", code);  
 return 0;  
 }  
   
 old\_stock = product->stock;  
 product->stock = new\_stock;  
   
 printf("商品「%s」の在庫を更新しました: %d個 → %d個\n",  
 product->name, old\_stock, new\_stock);  
   
 return 1;  
}  
  
/\* 在庫が少ない商品を表示する関数 \*/  
void display\_low\_stock(const struct Product products[], int count, int threshold)  
{  
 int i;  
 int found = 0;  
   
 printf("\n=== 在庫が%d個以下の商品 ===\n", threshold);  
   
 for (i = 0; i < count; i++) {  
 if (products[i].stock <= threshold) {  
 if (!found) {  
 printf("%-8s %-20s %8s\n", "コード", "商品名", "在庫");  
 printf("----------------------------------------\n");  
 found = 1;  
 }  
 printf("%-8d %-20s %8d個\n",  
 products[i].code,  
 products[i].name,  
 products[i].stock);  
 }  
 }  
   
 if (!found) {  
 printf("該当する商品はありません。\n");  
 }  
}  
  
/\* 価格帯で商品を検索する関数 \*/  
void search\_by\_price\_range(const struct Product products[], int count,  
 double min\_price, double max\_price)  
{  
 int i;  
 int found = 0;  
   
 printf("\n=== 価格 %.0f円 〜 %.0f円 の商品 ===\n", min\_price, max\_price);  
   
 for (i = 0; i < count; i++) {  
 if (products[i].price >= min\_price && products[i].price <= max\_price) {  
 if (!found) {  
 printf("%-8s %-20s %10s\n", "コード", "商品名", "価格");  
 printf("----------------------------------------\n");  
 found = 1;  
 }  
 printf("%-8d %-20s %10.0f円\n",  
 products[i].code,  
 products[i].name,  
 products[i].price);  
 }  
 }  
   
 if (!found) {  
 printf("該当する商品はありません。\n");  
 }  
}  
  
int main(void)  
{  
 struct Product products[MAX\_PRODUCTS];  
 int product\_count = 0;  
 int choice;  
   
 printf("===== 商品管理システム =====\n\n");  
   
 /\* デモ用: 初期データを追加 \*/  
 add\_product(products, &product\_count, 101, "ノートPC", 98000, 5);  
 add\_product(products, &product\_count, 102, "USBメモリ", 2500, 20);  
 add\_product(products, &product\_count, 103, "マウス", 1500, 15);  
 add\_product(products, &product\_count, 104, "キーボード", 3500, 8);  
 add\_product(products, &product\_count, 105, "モニター", 25000, 3);  
   
 /\* メニューのシミュレーション \*/  
 printf("\n=== 商品管理システム ===\n");  
 printf("1. 商品追加\n");  
 printf("2. 商品一覧表示\n");  
 printf("3. 在庫更新\n");  
 printf("0. 終了\n");  
   
 /\* デモ: 商品一覧を表示 \*/  
 printf("選択: 2\n");  
 display\_all\_products(products, product\_count);  
   
 /\* デモ: 新商品を追加 \*/  
 printf("\n選択: 1\n");  
 printf("\n商品コード: 106\n");  
 printf("商品名: ヘッドセット\n");  
 printf("価格: 4800\n");  
 printf("在庫数: 12\n");  
 add\_product(products, &product\_count, 106, "ヘッドセット", 4800, 12);  
   
 /\* デモ: 在庫更新 \*/  
 printf("\n選択: 3\n");  
 printf("\n更新する商品コード: 103\n");  
 printf("新しい在庫数: 25\n");  
 update\_stock(products, product\_count, 103, 25);  
   
 /\* 更新後の一覧表示 \*/  
 printf("\n選択: 2\n");  
 display\_all\_products(products, product\_count);  
   
 /\* 追加機能のデモ \*/  
 printf("\n\n=== 追加機能デモ ===\n");  
   
 /\* 在庫が少ない商品 \*/  
 display\_low\_stock(products, product\_count, 5);  
   
 /\* 価格帯検索 \*/  
 search\_by\_price\_range(products, product\_count, 2000, 5000);  
   
 /\* 商品統計 \*/  
 printf("\n\n=== 商品統計 ===\n");  
 if (product\_count > 0) {  
 int i;  
 double total\_value = 0.0;  
 double max\_price = products[0].price;  
 double min\_price = products[0].price;  
 int total\_stock = 0;  
 const char \*expensive\_product = products[0].name;  
 const char \*cheapest\_product = products[0].name;  
   
 for (i = 0; i < product\_count; i++) {  
 total\_value += products[i].price \* products[i].stock;  
 total\_stock += products[i].stock;  
   
 if (products[i].price > max\_price) {  
 max\_price = products[i].price;  
 expensive\_product = products[i].name;  
 }  
 if (products[i].price < min\_price) {  
 min\_price = products[i].price;  
 cheapest\_product = products[i].name;  
 }  
 }  
   
 printf("登録商品数: %d個\n", product\_count);  
 printf("在庫総数: %d個\n", total\_stock);  
 printf("在庫総額: %.0f円\n", total\_value);  
 printf("平均単価: %.0f円\n", total\_value / total\_stock);  
 printf("最高価格商品: %s (%.0f円)\n", expensive\_product, max\_price);  
 printf("最低価格商品: %s (%.0f円)\n", cheapest\_product, min\_price);  
 }  
   
 printf("\n選択: 0\n");  
 printf("システムを終了します。\n");  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 商品管理システム =====  
  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
  
=== 商品管理システム ===  
1. 商品追加  
2. 商品一覧表示  
3. 在庫更新  
0. 終了  
選択: 2  
  
=== 商品一覧 ===  
コード 商品名 価格 在庫  
--------------------------------------------------------  
101 ノートPC 98000円 5個  
102 USBメモリ 2500円 20個  
103 マウス 1500円 15個  
104 キーボード 3500円 8個  
105 モニター 25000円 3個  
--------------------------------------------------------  
登録商品数: 5個  
在庫総額: 693500円  
  
選択: 1  
  
商品コード: 106  
商品名: ヘッドセット  
価格: 4800  
在庫数: 12  
商品を追加しました。  
  
選択: 3  
  
更新する商品コード: 103  
新しい在庫数: 25  
商品「マウス」の在庫を更新しました: 15個 → 25個  
  
選択: 2  
  
=== 商品一覧 ===  
コード 商品名 価格 在庫  
--------------------------------------------------------  
101 ノートPC 98000円 5個  
102 USBメモリ 2500円 20個  
103 マウス 1500円 25個  
104 キーボード 3500円 8個  
105 モニター 25000円 3個  
106 ヘッドセット 4800円 12個  
--------------------------------------------------------  
登録商品数: 6個  
在庫総額: 766100円  
  
=== 追加機能デモ ===  
  
=== 在庫が5個以下の商品 ===  
コード 商品名 在庫  
----------------------------------------  
101 ノートPC 5個  
105 モニター 3個  
  
=== 価格 2000円 〜 5000円 の商品 ===  
コード 商品名 価格  
----------------------------------------  
102 USBメモリ 2500円  
104 キーボード 3500円  
106 ヘッドセット 4800円  
  
=== 商品統計 ===  
登録商品数: 6個  
在庫総数: 73個  
在庫総額: 766100円  
平均単価: 10495円  
最高価格商品: ノートPC (98000円)  
最低価格商品: マウス (1500円)  
  
選択: 0  
システムを終了します。  
\*/