C言語プログラミング教材 - 演習問題集（応用編）

第7章〜第11章の演習問題

C言語学習教材プロジェクト

2024年

Table of Contents

[はじめに 6](#_Toc202360962)

[前提知識 6](#_Toc202360963)

[演習問題の難易度 6](#_Toc202360964)

[ポインタ学習の重要性 6](#_Toc202360965)

[第7章: 配列（基本編） - 演習問題 7](#_Toc202360966)

[演習の目的 7](#_Toc202360967)

[基礎問題 7](#_Toc202360968)

[n 問題7-1: 配列の基本操作 7](#_Toc202360969)

[n 問題7-2: 配列の要素シフト 8](#_Toc202360970)

[n 問題7-3: 配列の統計処理 8](#_Toc202360971)

[応用問題 9](#_Toc202360972)

[n 問題7-4: 成績管理システム 9](#_Toc202360973)

[n 問題7-5: 行列演算 10](#_Toc202360974)

[n 問題7-6: 頻度分布表 11](#_Toc202360975)

[チャレンジ問題 12](#_Toc202360976)

[n 問題7-7: ソートアルゴリズムの実装 12](#_Toc202360977)

[n 問題7-8: ライフゲーム 13](#_Toc202360978)

[n 問題7-9: 魔方陣の生成と検証 14](#_Toc202360979)

[提出方法 15](#_Toc202360980)

[評価基準 15](#_Toc202360981)

[学習のポイント 15](#_Toc202360982)

[第8章: ポインタ基礎 - 演習問題 16](#_Toc202360983)

[基礎問題 16](#_Toc202360984)

[n 演習10-1: ポインタの基本操作 16](#_Toc202360985)

[n 演習10-2: 配列とポインタ 16](#_Toc202360986)

[n 演習10-3: 文字列操作 17](#_Toc202360987)

[応用問題 17](#_Toc202360988)

[n 演習10-4: 配列操作関数 17](#_Toc202360989)

[n 演習10-5: 文字列処理 18](#_Toc202360990)

[n 演習10-6: データ変換 19](#_Toc202360991)

[発展問題 19](#_Toc202360992)

[n 演習10-7: メモリ操作 19](#_Toc202360993)

[n 演習10-8: アルゴリズム実装 20](#_Toc202360994)

[n 演習10-9: データ構造 20](#_Toc202360995)

[チャレンジ問題 20](#_Toc202360996)

[n 演習10-10: 動的配列シミュレータ 20](#_Toc202360997)

[提出方法 21](#_Toc202360998)

[コンパイル例 21](#_Toc202360999)

[ヒント 21](#_Toc202361000)

[n ポインタの基本 21](#_Toc202361001)

[n 安全なプログラミング 21](#_Toc202361002)

[n デバッグのコツ 21](#_Toc202361003)

[n よくある間違い 22](#_Toc202361004)

[第9章: 配列とポインタ - 演習問題 23](#_Toc202361005)

[演習の目的 23](#_Toc202361006)

[基礎問題 23](#_Toc202361007)

[n 問題9-1: ポインタによる配列走査 23](#_Toc202361008)

[n 問題9-2: 関数への配列渡し 24](#_Toc202361009)

[応用問題 25](#_Toc202361010)

[n 問題9-3: 文字列のポインタ操作 25](#_Toc202361011)

[n 問題9-4: 多次元配列とポインタ 25](#_Toc202361012)

[n 問題9-5: 動的配列の模擬 26](#_Toc202361013)

[チャレンジ問題 27](#_Toc202361014)

[n 問題9-6: 高速文字列検索 27](#_Toc202361015)

[n 問題9-7: メモリプールの実装 28](#_Toc202361016)

[提出方法 29](#_Toc202361017)

[学習のポイント 29](#_Toc202361018)

[第10章: 文字列処理 - 演習問題 30](#_Toc202361019)

[演習の目的 30](#_Toc202361020)

[基礎問題 30](#_Toc202361021)

[n 問題10-1: 基本的な文字列操作 30](#_Toc202361022)

[n 問題10-2: 文字列の検索と置換 31](#_Toc202361023)

[応用問題 31](#_Toc202361024)

[n 問題10-3: 文字列配列の管理 31](#_Toc202361025)

[n 問題10-4: 文字列の分割と結合 32](#_Toc202361026)

[n 問題10-5: 文字列の変換と検証 33](#_Toc202361027)

[チャレンジ問題 34](#_Toc202361028)

[n 問題10-6: 高度な文字列検索 34](#_Toc202361029)

[n 問題10-7: テキストエディタの基本機能 35](#_Toc202361030)

[提出方法 36](#_Toc202361031)

[学習のポイント 36](#_Toc202361032)

[第11章: 関数 - 演習問題 37](#_Toc202361033)

[基礎問題 37](#_Toc202361034)

[n 問題11-1: 基本的な関数作成 37](#_Toc202361035)

[n 問題11-2: 配列操作関数 38](#_Toc202361036)

[n 問題11-3: 文字列処理関数 39](#_Toc202361037)

[応用問題 40](#_Toc202361038)

[n 問題11-4: 複数戻り値を持つ関数 40](#_Toc202361039)

[n 問題11-5: エラーハンドリング付き関数 41](#_Toc202361040)

[n 問題11-6: 構造体を使った関数 42](#_Toc202361041)

[チャレンジ問題 43](#_Toc202361042)

[n 問題11-7: 再帰関数 43](#_Toc202361043)

[n 問題11-8: 高度な文字列処理 44](#_Toc202361044)

[n 問題11-9: ソートアルゴリズム関数 44](#_Toc202361045)

[挑戦課題 45](#_Toc202361046)

[n 課題9-10: 関数ポインター 45](#_Toc202361047)

[n 課題9-11: メモリ効率を考慮した関数 45](#_Toc202361048)

[n 課題9-12: 総合プロジェクト 46](#_Toc202361049)

[提出要件 46](#_Toc202361050)

[評価ポイント 47](#_Toc202361051)

[難易度別の推奨学習順序 47](#_Toc202361052)

[n 初学者向け（基本課題から開始） 47](#_Toc202361053)

[n 中級者向け（中級課題に挑戦） 47](#_Toc202361054)

[n 上級者向け（上級・挑戦課題） 47](#_Toc202361055)

[学習のヒント 47](#_Toc202361056)

[参考資料 48](#_Toc202361057)

[実装例の構成 48](#_Toc202361058)

[発展的な学習内容 49](#_Toc202361059)

[演習問題を解き終えたら 49](#_Toc202361060)

[理解度の確認 49](#_Toc202361061)

[n ポインタの理解 49](#_Toc202361062)

[n 文字列処理 50](#_Toc202361063)

[n 関数設計 50](#_Toc202361064)

[次のステップ 50](#_Toc202361065)

[デバッグのヒント 50](#_Toc202361066)

# はじめに

本書は、C言語プログラミング教材応用編の演習問題集です。配列、ポインタ、文字列、関数など、C言語の中核となる概念を実践的に習得するための問題を収録しています。

## 前提知識

応用編の演習問題に取り組む前に、基礎編の内容を十分に理解していることが必要です：

* 基本的な文法とデータ型
* 演算子の使い方
* 条件分岐とループ

## 演習問題の難易度

応用編では、より実践的な問題を扱います：

* **基礎問題**: 新しい概念の基本的な使い方
* **応用問題**: 実務で遭遇する典型的な課題
* **発展問題**: アルゴリズムとデータ構造の応用

## ポインタ学習の重要性

特に第8章〜第9章のポインタ関連の演習は、C言語習得の要となります。じっくりと時間をかけて取り組んでください。

# 第7章: 配列（基本編） - 演習問題

## 演習の目的

* 1次元配列と多次元配列の理解を深める
* 配列を使った実践的な問題解決能力を養う
* 基本的な配列操作をマスターする

## 基礎問題

### 問題7-1: 配列の基本操作

整数配列に対する各種操作を実行するプログラムを作成してください。

**要件**

* 5個の整数を入力して配列に格納
* 以下の操作を実装:
  + 配列の全要素を表示
  + 最大値と最小値を見つける
  + 平均値を計算
  + 配列を逆順に並べ替える
  + 指定した値が配列内に存在するか検索

**実装のヒント**

* 配列のサイズは#defineで定義
* 逆順は別の配列にコピーまたはその場で交換

**期待される出力例:**

配列操作プログラム  
5個の整数を入力してください: 34 12 67 89 23  
  
配列の内容: 34 12 67 89 23  
最大値: 89  
最小値: 12  
平均値: 45.00  
  
逆順配列: 23 89 67 12 34  
  
検索する値を入力: 67  
値 67 は位置 2 に存在します（0から数えて）  
  
検索する値を入力: 50  
値 50 は配列内に存在しません

**ファイル名:** ex7\_1\_array\_operations.c

### 問題7-2: 配列の要素シフト

配列の要素を左右にシフトするプログラムを作成してください。

**要件**

* 10個の整数を配列に格納
* 以下の操作を実装:
  + 全要素を右に1つシフト（最後の要素は先頭へ）
  + 全要素を左に1つシフト（最初の要素は最後へ）
  + 指定した数だけシフト
  + シフト前後の配列を表示

**実装のヒント**

* 一時変数を使って端の要素を保存
* ループの方向に注意

**期待される出力例:**

配列シフトプログラム  
初期配列: 1 2 3 4 5 6 7 8 9 10  
  
右に1シフト: 10 1 2 3 4 5 6 7 8 9  
左に1シフト: 2 3 4 5 6 7 8 9 10 1  
  
シフト数を入力してください: 3  
右に3シフト: 8 9 10 1 2 3 4 5 6 7  
  
シフト数を入力してください: -2  
左に2シフト: 3 4 5 6 7 8 9 10 1 2

**ファイル名:** ex7\_2\_array\_shift.c

### 問題7-3: 配列の統計処理

整数配列に対する基本的な統計処理を行うプログラムを作成してください。

**要件**

* 10個の整数データを配列に格納
* 以下の統計値を計算:
  + 合計値と平均値
  + 最大値と最小値およびその位置
  + 標準偏差（簡易計算でよい）

**実装のヒント**

* 標準偏差は各要素と平均の差の二乗の平均の平方根

**期待される出力例:**

配列統計処理プログラム  
10個の整数を入力してください: 78 65 92 83 71 96 54 88 67 79  
  
=== 統計情報 ===  
データ: 78 65 92 83 71 96 54 88 67 79  
合計値: 773  
平均値: 77.30  
最大値: 96 (位置: 5)  
最小値: 54 (位置: 6)  
標準偏差: 12.85

**ファイル名:** ex7\_3\_array\_statistics.c

## 応用問題

### 問題7-4: 成績管理システム

複数の学生の複数科目の成績を管理するプログラムを作成してください。

**要件**

* 5人の学生、4科目（国語、数学、英語、理科）の成績を2次元配列で管理
* 各学生の合計点と平均点を計算
* 各科目の平均点を計算
* 最高得点の学生と科目を表示
* 成績表を見やすく表示

**実装のヒント**

* 2次元配列 grades[][] を使用
* 行が学生、列が科目

**期待される出力例:**

成績管理システム  
  
成績表:  
学生 国語 数学 英語 理科 合計 平均  
-------------------------------------------  
学生1 85 92 78 88 343 85.8  
学生2 76 88 91 83 338 84.5  
学生3 92 76 85 90 343 85.8  
学生4 68 95 73 79 315 78.8  
学生5 89 84 88 86 347 86.8  
-------------------------------------------  
科目平均 82.0 87.0 83.0 85.2  
  
最高得点: 学生4の数学 95点  
クラス全体の平均点: 84.3

**ファイル名:** ex7\_4\_grade\_management.c

### 問題7-5: 行列演算

2次元配列を使って行列演算を実行するプログラムを作成してください。

**要件**

* 3x3の行列を2つ入力
* 以下の演算を実装:
  + 行列の加算
  + 行列の減算
  + 行列の乗算
  + 転置行列の計算

**実装のヒント**

* 行列の乗算: result[i][j] = Σ(a[i][k] \* b[k][j])
* 各演算結果を見やすく表示

**期待される出力例:**

行列演算プログラム  
  
行列A:  
1 2 3  
4 5 6  
7 8 9  
  
行列B:  
9 8 7  
6 5 4  
3 2 1  
  
加算結果 (A + B):  
10 10 10  
10 10 10  
10 10 10  
  
減算結果 (A - B):  
-8 -6 -4  
-2 0 2  
 4 6 8  
  
乗算結果 (A × B):  
30 24 18  
84 69 54  
138 114 90  
  
転置行列 (A^T):  
1 4 7  
2 5 8  
3 6 9

**ファイル名:** ex7\_5\_matrix\_operations.c

### 問題7-6: 頻度分布表

テストの点数分布を分析するプログラムを作成してください。

**要件**

* 30人分のテスト点数（0-100点）を配列に格納
* 10点刻みで頻度分布表を作成
* ヒストグラムを\*を使って表示
* 最頻値の範囲を表示

**出力例:**

0- 9: \*\* (2人)  
10-19: \*\*\* (3人)  
20-29: \* (1人)  
...  
90-99: \*\*\*\*\*\*\*\* (8人)

**ファイル名:** ex7\_6\_frequency\_distribution.c

## チャレンジ問題

### 問題7-7: ソートアルゴリズムの実装

配列のソートアルゴリズムを複数実装して比較してください。

**要件**

* 以下のソートアルゴリズムを実装:
  + バブルソート
  + 選択ソート
  + 挿入ソート
* 各アルゴリズムの実行時間を測定
* ランダムな配列、ソート済み配列、逆順配列で性能比較

**追加要件**

* 比較回数と交換回数をカウント
* 結果をグラフィカルに表示（\*を使った簡易グラフ）

**期待される出力例:**

ソートアルゴリズムの比較  
  
配列サイズ: 20  
元の配列: 34 7 23 32 5 62 78 4 87 12 45 28 91 15 53 67 39 22 84 10  
  
=== バブルソート ===  
ソート後: 4 5 7 10 12 15 22 23 28 32 34 39 45 53 62 67 78 84 87 91  
比較回数: 190  
交換回数: 92  
実行時間: 0.002ms  
  
=== 選択ソート ===  
ソート後: 4 5 7 10 12 15 22 23 28 32 34 39 45 53 62 67 78 84 87 91  
比較回数: 190  
交換回数: 19  
実行時間: 0.001ms  
  
=== 挿入ソート ===  
ソート後: 4 5 7 10 12 15 22 23 28 32 34 39 45 53 62 67 78 84 87 91  
比較回数: 92  
交換回数: 92  
実行時間: 0.001ms  
  
効率性比較グラフ（比較回数）:  
バブル: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (190)  
選択 : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (190)  
挿入 : \*\*\*\*\*\*\*\*\*\* (92)

**ファイル名:** ex7\_7\_sort\_algorithms.c

### 問題7-8: ライフゲーム

コンウェイのライフゲームの簡易版を実装してください。

**要件**

* 20x20の2次元配列でセルの状態を管理
* セルは生（1）または死（0）の状態
* 各世代で以下のルールを適用:
  + 生きているセルの周囲に2-3個の生きたセルがある→生存
  + 死んでいるセルの周囲にちょうど3個の生きたセルがある→誕生
  + それ以外→死亡
* 初期状態をランダムまたはパターンで設定
* 各世代の状態を表示

**期待される出力例:**

コンウェイのライフゲーム（20x20）  
  
世代 0:  
....................  
......\*\*\*.........  
......\*..\*........  
.......\*..........  
....................  
（以下省略）  
  
世代 1:  
....................  
.......\*..........  
......\*..\*........  
......\*...........  
....................  
（以下省略）  
  
世代 2:  
....................  
......\*\*\*.........  
......\*..\*........  
.......\*..........  
....................  
（パターンが繰り返される）  
  
10世代後の生存セル数: 8

**ファイル名:** ex7\_8\_life\_game.c

### 問題7-9: 魔方陣の生成と検証

魔方陣を生成・検証するプログラムを作成してください。

**要件**

* 3x3の魔方陣を生成（1-9の数字を使用）
* 既存の配列が魔方陣かどうかを検証
* 検証条件:
  + 各行の合計が等しい
  + 各列の合計が等しい
  + 対角線の合計が等しい
  + 1-9の数字が重複なく使用されている
* 複数の魔方陣パターンを生成可能

**期待される出力例:**

魔方陣生成・検証プログラム  
  
生成された魔方陣:  
2 7 6  
9 5 1  
4 3 8  
  
検証結果:  
行の合計: 15, 15, 15 ✓  
列の合計: 15, 15, 15 ✓  
対角線の合計: 15, 15 ✓  
すべての数字が1-9で重複なし ✓  
  
これは正しい魔方陣です！  
  
別の魔方陣パターン:  
8 1 6  
3 5 7  
4 9 2  
  
検証結果: 正しい魔方陣です！

**ファイル名:** ex7\_9\_magic\_square.c

## 提出方法

1. 各問題に対して個別のCファイルを作成
2. ファイル名は指定された名前を使用
3. コメントで問題番号と簡単な説明を記載
4. コンパイル・実行確認を必ず行う

## 評価基準

* **正確性**: 要求仕様を満たしているか
* **効率性**: 配列操作の最適化
* **可読性**: 適切なインデント、変数名、コメント
* **境界処理**: 配列の境界チェック

## 学習のポイント

* 配列の初期化と要素アクセス
* 多次元配列の使い方
* 配列を使った効率的なデータ処理
* 配列の境界を超えないための注意点

# 第8章: ポインタ基礎 - 演習問題

## 基礎問題

### 演習10-1: ポインタの基本操作

2つの整数変数の値をポインタを使って交換するプログラムを作成してください。

**要件**

* 2つの整数変数を宣言・初期化
* ポインタを使った値の交換関数を実装
* 交換前後の値とアドレスを表示
* 直接的な値の交換とポインタ経由の交換を比較

**期待される出力例:**

交換前: a = 10 (アドレス: 0x7fffbffac), b = 20 (アドレス: 0x7fffbffa8)  
ポインタを使った交換実行...  
交換後: a = 20 (アドレス: 0x7fffbffac), b = 10 (アドレス: 0x7fffbffa8)

### 演習10-2: 配列とポインタ

ポインタ演算を使って配列の要素を逆順に表示するプログラムを作成してください。

**要件**

* 整数配列を宣言・初期化
* ポインタ演算で配列の最後から最初に向かってアクセス
* インデックス記法とポインタ記法の両方で表示
* 配列のサイズを動的に計算

**期待される出力例:**

配列の内容（順方向）:  
インデックス記法: arr[0]=1, arr[1]=2, arr[2]=3, arr[3]=4, arr[4]=5  
ポインタ記法: \*ptr=1, \*(ptr+1)=2, \*(ptr+2)=3, \*(ptr+3)=4, \*(ptr+4)=5  
  
配列の内容（逆順）:  
ポインタ演算で逆順表示: 5 4 3 2 1  
  
各要素のアドレス:  
&arr[0] = 0x7fffc000  
&arr[1] = 0x7fffc004  
&arr[2] = 0x7fffc008  
&arr[3] = 0x7fffc00c  
&arr[4] = 0x7fffc010

### 演習10-3: 文字列操作

ポインタを使って文字列の長さを計算し、文字列を逆順にするプログラムを作成してください。

**要件**

* 文字列の長さ計算関数（strlen相当）をポインタで実装
* 文字列を逆順にする関数をポインタで実装
* 元の文字列と逆順文字列を表示
* 文字列リテラルと文字配列の違いを考慮

**期待される出力例:**

文字列操作プログラム  
文字列を入力: Hello World  
  
元の文字列: Hello World  
文字列の長さ: 11文字  
  
文字列を逆順に変換中...  
逆順文字列: dlroW olleH  
  
各文字とそのアドレス:  
'H' at 0x7fffc100  
'e' at 0x7fffc101  
'l' at 0x7fffc102  
... (以下省略)

## 応用問題

### 演習10-4: 配列操作関数

ポインタを使って配列の最大値、最小値、平均値を計算する関数群を作成してください。

**要件**

* 最大値を見つけてそのポインタを返す関数
* 最小値を見つけてそのポインタを返す関数
* 平均値を計算する関数（戻り値はdouble）
* 各関数の結果を使ってレポートを作成

**実装する関数:**

int\* find\_max(int \*arr, int size);  
int\* find\_min(int \*arr, int size);  
double calculate\_average(int \*arr, int size);  
void print\_statistics(int \*arr, int size);

**期待される出力例:**

配列統計分析プログラム  
配列: 34 12 67 89 23 45 78 56 90 11  
  
=== 統計情報 ===  
最大値: 90 (アドレス: 0x7fffc120, インデックス: 8)  
最小値: 11 (アドレス: 0x7fffc124, インデックス: 9)  
平均値: 50.50  
  
値の範囲: 11 〜 90

### 演習10-5: 文字列処理

ポインタを使って文字列の検索、置換、分割を実行する関数群を作成してください。

**要件**

* 文字列内で特定の文字を検索する関数
* 文字列内の文字を置換する関数
* 文字列を特定の文字で分割する関数
* 大文字・小文字を無視した比較機能

**期待される出力例:**

文字列処理プログラム  
文字列: Hello,World,Programming  
  
文字 'o' の検索:  
見つかった位置: 4, 7, 18  
  
文字 'o' を 'O' に置換:  
置換後: HellO,WOrld,PrOgramming  
  
カンマで分割:  
1: Hello  
2: World  
3: Programming  
  
大文字小文字を無視した検索 ('w'):  
見つかった位置: 6 (W)

### 演習10-6: データ変換

ポインタを使って配列のデータ型変換を実行するプログラムを作成してください。

**要件**

* int配列をfloat配列に変換
* 文字列を数値配列に変換
* バイト配列を整数として解釈
* エラーハンドリングの実装

**期待される出力例:**

データ型変換プログラム  
  
=== int配列からfloat配列への変換 ===  
元のint配列: 10, 20, 30, 40, 50  
変換後のfloat配列: 10.00, 20.00, 30.00, 40.00, 50.00  
  
=== 文字列から数値配列への変換 ===  
文字列: "12 34 56 78 90"  
数値配列: 12, 34, 56, 78, 90  
  
=== バイト配列を整数として解釈 ===  
バイト配列: 0x01 0x02 0x03 0x04  
リトルエンディアン解釈: 67305985 (0x04030201)  
ビッグエンディアン解釈: 16909060 (0x01020304)

## 発展問題

### 演習10-7: メモリ操作

ポインタを使って任意のデータ型の配列をコピーする汎用関数を作成してください。

**要件**

* void\*ポインタを使った汎用コピー関数
* バイト単位でのメモリコピー
* 型安全性の考慮
* コピー対象の境界チェック

**関数プロトタイプ:**

void\* generic\_copy(void \*dest, const void \*src, size\_t size);  
int compare\_memory(const void \*ptr, const void \*ptr, size\_t size);

### 演習10-8: アルゴリズム実装

ポインタを使って各種ソートアルゴリズムを実装してください。

**要件**

* バブルソート（ポインタ版）
* 選択ソート（ポインタ版）
* 挿入ソート（ポインタ版）
* 汎用的な比較関数の使用

### 演習10-9: データ構造

ポインタを使って簡単なリンクリスト構造を実装してください。

**要件**

* ノード構造体の定義
* リストへの要素追加・削除
* リストの走査・検索
* メモリリークの防止

**構造体例:**

typedef struct Node {  
 int data;  
 struct Node \*next;  
} Node;

## チャレンジ問題

### 演習10-10: 動的配列シミュレータ

malloc/freeを使わずに、静的配列とポインタを使って動的配列のような動作を実現してください。

**要件**

* 固定サイズの大きな配列をメモリプールとして使用
* 要素の追加・削除・挿入機能
* メモリの断片化管理
* ガベージコレクション機能

## 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 ex\_pointer\_swap.c, ex\_array\_reverse.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 ex\_pointer\_swap.c -o ex\_pointer\_swap  
./ex\_pointer\_swap

## ヒント

### ポインタの基本

* 宣言: int \*ptr;
* アドレス取得: ptr = &variable;
* 値の取得: value = \*ptr;
* 演算: ptr + 1 は次の要素を指す

### 安全なプログラミング

* NULLポインタのチェック: if (ptr != NULL)
* 配列の境界チェック
* 初期化されていないポインタの使用を避ける

### デバッグのコツ

* アドレスと値を分けて表示
* ポインタ演算の結果を段階的に確認
* コンパイラの警告に注意を払う

### よくある間違い

* \*ptr++ と (\*ptr)++ の違い
* ローカル変数のアドレスを返すこと
* 配列の範囲外アクセス
* NULLポインタの参照

# 第9章: 配列とポインタ - 演習問題

## 演習の目的

* 配列とポインタの関係を深く理解する
* ポインタ演算を使った配列操作をマスターする
* 関数への配列の渡し方を習得する
* より効率的な配列処理技術を身につける

## 基礎問題

### 問題9-1: ポインタによる配列走査

ポインタを使って配列を操作するプログラムを作成してください。

**要件**

* 10個の整数を配列に格納
* 以下の操作をポインタを使って実装:
  + 配列の全要素を表示（配列記法を使わない）
  + 最大値と最小値を見つける（ポインタ演算使用）
  + 配列を逆順に並べ替える（ポインタで実装）
  + 特定の値を検索し、そのアドレスを返す

**実装のヒント**

* arr[i] の代わりに \*(arr + i) を使用
* ポインタのインクリメント/デクリメントを活用

**期待される出力例:**

=== ポインタによる配列操作 ===  
配列の内容: 34 67 12 89 23 45 78 56 90 11  
  
ポインタ記法での表示:  
\*(ptr+0) = 34  
\*(ptr+1) = 67  
\*(ptr+2) = 12  
... (続く)  
  
最大値: 90 (アドレス: 0x7fffc120)  
最小値: 11 (アドレス: 0x7fffc124)  
  
配列を逆順に並べ替え中...  
逆順配列: 11 90 56 78 45 23 89 12 67 34  
  
値 45 を検索...  
値 45 は アドレス 0x7fffc114 に見つかりました（インデックス: 5）

### 問題9-2: 関数への配列渡し

配列を関数に渡して操作するプログラムを作成してください。

**要件**

* 配列操作を行う以下の関数を実装:
  + 配列の要素をすべてn倍する関数
  + 2つの配列を要素ごとに加算する関数
  + 配列内の最大値へのポインタを返す関数
  + 配列を指定位置で分割する関数

**実装のヒント**

* 関数の引数は int \*arr または int arr[]
* サイズ情報は別引数で渡す

**期待される出力例:**

=== 関数への配列渡し ===  
元の配列: 1 2 3 4 5  
  
配列の要素を3倍にする:  
結果: 3 6 9 12 15  
  
配列A: 10 20 30 40 50  
配列B: 5 10 15 20 25  
要素ごとの加算結果: 15 30 45 60 75  
  
配列内の最大値: 75 (アドレス: 0x7fffc1a4)  
  
配列を位置3で分割:  
前半: 15 30 45  
後半: 60 75

## 応用問題

### 問題9-3: 文字列のポインタ操作

文字列をポインタで操作するプログラムを作成してください。

**要件**

* ポインタを使った以下の文字列関数を実装:
  + 文字列の長さを計算（strlen相当）
  + 文字列をコピー（strcpy相当）
  + 文字列を連結（strcat相当）
  + 文字列を反転

**実装のヒント**

* char \*を使った操作
* null終端文字の処理に注意

**期待される出力例:**

=== 文字列のポインタ操作 ===  
文字列1: Hello  
文字列2: World  
  
my\_strlen("Hello") = 5  
  
文字列コピー実行...  
コピー結果: Hello  
  
文字列連結実行...  
連結結果: HelloWorld  
  
文字列反転実行...  
反転結果: dlroWolleH

### 問題9-4: 多次元配列とポインタ

2次元配列をポインタで操作するプログラムを作成してください。

**要件**

* 3×3の行列に対する以下の操作:
  + ポインタを使った要素アクセス
  + 行列の転置（ポインタ使用）
  + 対角要素の合計
  + 行ごと、列ごとの合計

**実装のヒント**

* \*(\*(arr + i) + j) でアクセス
* int (\*ptr)[3] の使い方を理解

**期待される出力例:**

=== 多次元配列とポインタ ===  
元の行列:  
1 2 3  
4 5 6  
7 8 9  
  
ポインタを使った要素アクセス:  
\*(\*(matrix+1)+1) = 5  
  
転置行列:  
1 4 7  
2 5 8  
3 6 9  
  
対角要素の合計: 15 (1 + 5 + 9)  
  
行ごとの合計: 6, 15, 24  
列ごとの合計: 12, 15, 18

### 問題9-5: 動的配列の模擬

ポインタを使った配列の動的な操作を行うプログラムを作成してください。

**要件**

* 固定サイズ配列を使いながら動的配列のような操作:
  + 使用中の要素数を管理
  + 要素の追加（容量チェック付き）
  + 要素の削除（詰める処理）
  + 配列の拡張（別の大きな配列へコピー）

**実装のヒント**

* 実際の要素数と配列容量を別管理
* ポインタを使った効率的なコピー

**期待される出力例:**

=== 動的配列の模擬 ===  
初期容量: 10, 使用中: 0  
  
要素を追加: 10, 20, 30  
現在の配列: [10, 20, 30]  
容量: 10, 使用中: 3  
  
要素 20 を削除...  
現在の配列: [10, 30]  
容量: 10, 使用中: 2  
  
要素を追加して容量オーバー時の処理:  
新しい容量: 20  
配列の内容をコピー中...  
拡張完了！

## チャレンジ問題

### 問題9-6: 高速文字列検索

ポインタを使った効率的な文字列検索アルゴリズムを実装してください。

**要件**

* Boyer-Moore法の簡易版を実装:
  + 文字列内から部分文字列を検索
  + 見つかった位置のポインタを返す
  + 複数箇所で見つかる場合は全て表示

**追加要件**

* 検索回数をカウントして効率性を評価
* 通常の逐次検索と比較

**期待される出力例:**

=== 高速文字列検索 ===  
テキスト: "The quick brown fox jumps over the lazy dog"  
検索パターン: "the"  
  
Boyer-Moore法（簡易版）:  
見つかった位置: 31 (0x7fffc200)  
検索回数: 12回  
  
通常の逐次検索:  
見つかった位置: 31  
検索回数: 32回  
  
効率性向上: 62.5%

### 問題9-7: メモリプールの実装

大きな配列を使ったメモリプールを実装してください。

**要件**

* 大きな char配列をメモリプールとして使用
* 以下の機能を実装:
  + 指定サイズのメモリ割り当て
  + メモリの解放
  + 使用状況の表示
  + フラグメンテーションの管理

**実装のヒント**

* ポインタ演算でメモリブロックを管理
* 各ブロックの使用状況をヘッダで管理

**期待される出力例:**

=== メモリプール管理 ===  
プールサイズ: 1024 バイト  
  
メモリ割り当て:  
ブロック1: 100バイト確保 (アドレス: 0x7fffc000)  
ブロック2: 200バイト確保 (アドレス: 0x7fffc064)  
ブロック3: 150バイト確保 (アドレス: 0x7fffc12c)  
  
使用状況:  
使用中: 450バイト (43.9%)  
空き: 574バイト (56.1%)  
  
ブロック2を解放...  
フラグメンテーション発生: 3個の断片  
  
デフラグ実行...  
連続した空き領域: 774バイト

## 提出方法

1. 各演習問題ごとに独立したCファイルを作成
2. ファイル名は ex9\_1.c, ex9\_2.c のような形式で
3. ポインタの使用を明確にコメントで説明
4. 境界チェックを忘れずに実装

## 学習のポイント

* 配列記法とポインタ記法の変換を意識
* ポインタ演算の正確な理解
* 関数への配列渡しの仕組みを完全に理解
* メモリレイアウトを常に意識して実装

# 第10章: 文字列処理 - 演習問題

## 演習の目的

* C言語の文字列（文字配列）の仕組みを深く理解する
* 標準ライブラリの文字列関数を使いこなす
* 安全な文字列操作の実装方法を習得する
* 文字列処理アルゴリズムを実装する

## 基礎問題

### 問題10-1: 基本的な文字列操作

標準ライブラリの関数を使わずに、基本的な文字列操作関数を実装してください。

**要件**

* 以下の関数を実装:
  + my\_strlen() - 文字列の長さを計算
  + my\_strcpy() - 文字列をコピー
  + my\_strcmp() - 文字列を比較
  + my\_strcat() - 文字列を連結

**実装のヒント**

* null終端文字(‘\0’)の扱いに注意
* バッファオーバーフローを防ぐ

**期待される出力例:**

=== 基本的な文字列操作 ===  
文字列1: "Hello"  
文字列2: "World"  
  
my\_strlen("Hello") = 5  
my\_strlen("World") = 5  
  
my\_strcpy実行後: "Hello"  
my\_strcmp("Hello", "World") = -15 (Hello < World)  
my\_strcmp("Hello", "Hello") = 0 (等しい)  
  
my\_strcat実行後: "HelloWorld"

### 問題10-2: 文字列の検索と置換

文字列内の文字や部分文字列を検索・置換する関数を実装してください。

**要件**

* 以下の関数を実装:
  + my\_strchr() - 文字を検索
  + my\_strrchr() - 文字を後方検索
  + 特定文字の出現回数をカウント
  + 文字列内の特定文字を別の文字に置換

**実装のヒント**

* ポインタを使った効率的な実装
* NULLチェックを忘れずに

**期待される出力例:**

=== 文字列の検索と置換 ===  
文字列: "Hello, World!"  
  
文字 'o' の検索:  
my\_strchr: 最初の 'o' は位置 4  
my\_strrchr: 最後の 'o' は位置 7  
出現回数: 2回  
  
文字 'l' を 'L' に置換:  
置換後: "HeLLo, WorLd!"

## 応用問題

### 問題10-3: 文字列配列の管理

学生名を格納する文字列配列の管理システムを実装してください。

**要件**

* 最大20人の学生名を管理（各名前は最大50文字）
* 以下の機能を実装:
  + 学生名の追加
  + 学生名の削除
  + 学生名の検索
  + アルファベット順にソート
  + 全学生名の表示

**実装のヒント**

* 二次元配列を使用
* strcmp()でソートを実装

**期待される出力例:**

=== 学生名管理システム ===  
1. 学生追加  
2. 学生削除  
3. 学生検索  
4. アルファベット順ソート  
5. 全学生表示  
6. 終了  
  
選択: 1  
学生名: Tanaka  
追加しました。  
  
選択: 1  
学生名: Sato  
追加しました。  
  
選択: 5  
--- 学生一覧 ---  
1. Tanaka  
2. Sato  
  
選択: 4  
ソート完了！  
  
選択: 5  
--- 学生一覧 ---  
1. Sato  
2. Tanaka

### 問題10-4: 文字列の分割と結合

文字列を特定のデリミタで分割・結合する関数を実装してください。

**要件**

* カンマ区切りの文字列を分割して配列に格納
* 配列の文字列を指定のデリミタで結合
* 空白文字（スペース、タブ、改行）で分割
* トークンの前後の空白を除去（トリム機能）

**実装のヒント**

* strtok()の仕組みを理解
* 動的な配列管理を考慮

**期待される出力例:**

=== 文字列の分割と結合 ===  
元の文字列: "apple,banana,orange,grape"  
  
カンマで分割:  
1: "apple"  
2: "banana"  
3: "orange"  
4: "grape"  
  
スペースで結合: "apple banana orange grape"  
  
空白文字で分割（トリム付き）:  
元の文字列: " hello world \t programming \n"  
1: "hello"  
2: "world"  
3: "programming"

### 問題10-5: 文字列の変換と検証

文字列の変換および検証を行う関数を実装してください。

**要件**

* 文字列を大文字/小文字に変換
* 文字列が数値として有効か検証
* 文字列を整数/浮動小数点数に変換
* パスワードの強度チェック（英数字、記号を含む8文字以上）

**実装のヒント**

* ctype.h の関数を活用
* エラーハンドリングを適切に

**期待される出力例:**

=== 文字列の変換と検証 ===  
文字列: "Hello World 123"  
大文字変換: "HELLO WORLD 123"  
小文字変換: "hello world 123"  
  
数値検証:  
"123" → 有効な整数  
"12.34" → 有効な実数  
"abc123" → 無効な数値  
  
文字列を数値に変換:  
"123" → 123 (整数)  
"45.67" → 45.67 (実数)  
  
パスワード強度チェック:  
"password" → 弱い（英字のみ）  
"Pass123!" → 強い（英数字と記号を含む8文字）

## チャレンジ問題

### 問題10-6: 高度な文字列検索

パターンマッチングアルゴリズムを実装してください。

**要件**

* ワイルドカード検索（\*、?）の実装
* 大小文字を区別しない検索
* 単語境界での検索（単語単位の検索）
* 複数パターンの同時検索

**追加要件**

* 検索アルゴリズムの効率性を考慮
* KMP法やBoyer-Moore法の簡易版を実装

**期待される出力例:**

=== 高度な文字列検索 ===  
テキスト: "The quick brown fox jumps over the lazy dog"  
  
ワイルドカード検索:  
パターン "\*fox\*" → マッチ（位置: 16）  
パターン "qu?ck" → マッチ（位置: 4）  
  
大小文字を区別しない検索:  
"THE" → 見つかった位置: 0, 31  
  
単語境界検索:  
"the" → 見つかった単語位置: 31（"The"は単語境界でマッチしない）  
  
複数パターン同時検索:  
パターン: ["quick", "fox", "dog"]  
結果:  
- "quick": 位置 4  
- "fox": 位置 16  
- "dog": 位置 40

### 問題10-7: テキストエディタの基本機能

簡単なテキストエディタの基本機能を実装してください。

**要件**

* 複数行のテキストを管理（最大100行、各行最大80文字）
* 以下の機能を実装:
  + 行の挿入・削除
  + 文字列の検索・置換（全置換機能付き）
  + 行番号付きの表示
  + ファイルへの保存と読み込み

**実装のヒント**

* 二次元配列でテキストを管理
* ファイルI/Oの適切なエラー処理

**期待される出力例:**

=== 簡易テキストエディタ ===  
1. 行の挿入  
2. 行の削除  
3. 検索  
4. 置換  
5. 表示  
6. 保存  
7. 読み込み  
8. 終了  
  
選択: 1  
行番号: 1  
内容: Hello, World!  
挿入しました。  
  
選択: 5  
1: Hello, World!  
  
選択: 3  
検索文字列: World  
見つかりました: 行1, 位置7  
  
選択: 4  
検索文字列: World  
置換文字列: Programming  
全置換しますか？(y/n): y  
1件置換しました。  
  
選択: 5  
1: Hello, Programming!

## 提出方法

1. 各演習問題ごとに独立したCファイルを作成
2. ファイル名は ex10\_1.c, ex10\_2.c のような形式で
3. 安全な文字列操作を心がけ、バッファオーバーフローに注意
4. エラーハンドリングを適切に実装

## 学習のポイント

* null終端文字列の概念を確実に理解
* バッファサイズと文字列長の違いを意識
* ポインタと配列の関係を文字列で実践
* セキュアな文字列操作の重要性を認識

# 第11章: 関数 - 演習問題

## 基礎問題

### 問題11-1: 基本的な関数作成

以下の関数を作成してください。

1. **数学関数**
   * 2つの整数の最大値を返す関数 int max(int a, int b)
   * 3つの整数の最小値を返す関数 int min(int a, int b, int c)
   * 数値が偶数かどうかを判定する関数 int is\_even(int n)
   * 数値が素数かどうかを判定する関数 int is\_prime(int n)
2. **文字・文字列関数**
   * 文字が英字かどうかを判定する関数 int is\_letter(char c)
   * 文字列の長さを計算する関数 int my\_strlen(const char str[])
   * 文字列内の特定文字を数える関数 int count\_char(const char str[], char ch)
3. **表示関数**
   * 指定した長さの線を描く関数 void draw\_line(int length, char ch)
   * 数値を指定した桁数で表示する関数 void print\_number\_padded(int num, int width)

**期待される出力例:**

=== 数学関数のテスト ===  
max(10, 20) = 20  
min(30, 15, 25) = 15  
is\_even(24) = 1 (偶数)  
is\_even(17) = 0 (奇数)  
is\_prime(17) = 1 (素数)  
is\_prime(24) = 0 (素数ではない)  
  
=== 文字・文字列関数のテスト ===  
is\_letter('A') = 1  
is\_letter('5') = 0  
my\_strlen("Hello") = 5  
count\_char("Hello World", 'l') = 3  
  
=== 表示関数のテスト ===  
draw\_line(20, '-'): --------------------  
print\_number\_padded(42, 5): 00042  
print\_number\_padded(1234, 3): 1234

### 問題11-2: 配列操作関数

以下の配列操作関数を作成してください。

1. **基本操作**
   * 配列の要素の合計を計算する関数 int array\_sum(int arr[], int size)
   * 配列の平均値を計算する関数 double array\_average(int arr[], int size)
   * 配列内の最大値のインデックスを返す関数 int find\_max\_index(int arr[], int size)
2. **検索・ソート**
   * 線形検索関数 int linear\_search(int arr[], int size, int target)
   * 配列を昇順にソートする関数 void sort\_array(int arr[], int size)
   * 配列の要素を逆順にする関数 void reverse\_array(int arr[], int size)
3. **配列統計**
   * 配列の最大値と最小値を同時に求める関数 void find\_min\_max(int arr[], int size, int \*min, int \*max)
   * 配列内の重複する要素の数を数える関数 int count\_duplicates(int arr[], int size)

**期待される出力例:**

=== 配列操作関数のテスト ===  
配列: [34, 12, 67, 89, 23, 45, 78, 56, 90, 11]  
  
基本操作:  
array\_sum() = 495  
array\_average() = 49.50  
find\_max\_index() = 8 (値: 90)  
  
検索・ソート:  
linear\_search(45) = 5  
ソート前: [34, 12, 67, 89, 23, 45, 78, 56, 90, 11]  
ソート後: [11, 12, 23, 34, 45, 56, 67, 78, 89, 90]  
逆順: [90, 89, 78, 67, 56, 45, 34, 23, 12, 11]  
  
配列統計:  
最小値: 11, 最大値: 90  
重複要素数: 0  
  
配列: [1, 2, 3, 2, 4, 3, 5]  
重複要素数: 2 (2と3が重複)

### 問題11-3: 文字列処理関数

以下の文字列処理関数を作成してください。

1. **基本処理**
   * 文字列をコピーする関数 void my\_strcpy(char dest[], const char src[])
   * 文字列を連結する関数 void my\_strcat(char dest[], const char src[])
   * 文字列を比較する関数 int my\_strcmp(const char str1[], const char str2[])
2. **変換処理**
   * 文字列を大文字に変換する関数 void to\_uppercase(char str[])
   * 文字列を小文字に変換する関数 void to\_lowercase(char str[])
   * 文字列を逆順にする関数 void reverse\_string(char str[])
3. **解析処理**
   * 文字列内の単語数を数える関数 int count\_words(const char str[])
   * 文字列が回文かどうかを判定する関数 int is\_palindrome(const char str[])
   * 文字列内の母音の数を数える関数 int count\_vowels(const char str[])

**期待される出力例:**

=== 文字列処理関数のテスト ===  
  
基本処理:  
元の文字列: "Hello"  
my\_strcpy後: "Hello"  
my\_strcat後: "HelloWorld"  
my\_strcmp("Hello", "World") = -15  
  
変換処理:  
元: "Hello World"  
大文字: "HELLO WORLD"  
小文字: "hello world"  
逆順: "dlroW olleH"  
  
解析処理:  
"Hello World Programming" の単語数: 3  
"racecar" は回文: Yes  
"hello" は回文: No  
"Hello World" の母音数: 3 (e, o, o)

## 応用問題

### 問題11-4: 複数戻り値を持つ関数

以下の関数を作成してください。

1. **時間計算**
   * 秒数を時分秒に変換する関数 void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds)
   * 日数から年月日に変換する関数（365日/年として） void days\_to\_ymd(int total\_days, int \*years, int \*months, int \*days)
2. **座標計算**
   * 2点間の距離と中点を計算する関数 void calculate\_line\_info(double x1, double y1, double x2, double y2, double \*distance, double \*mid\_x, double \*mid\_y)
   * 円の面積と周囲の長さを計算する関数 void calculate\_circle\_info(double radius, double \*area, double \*circumference)
3. **統計計算**
   * 配列の統計情報を計算する関数 void calculate\_statistics(int arr[], int size, int \*min, int \*max, double \*mean, double \*median)
   * 成績から評価を計算する関数 void calculate\_grade\_info(int scores[], int count, double \*average, char \*grade, int \*pass\_count)

**期待される出力例:**

=== 複数戻り値関数のテスト ===  
  
時間計算:  
3661秒 = 1時間 1分 1秒  
400日 = 1年 1月 5日  
  
座標計算:  
点(0, 0)と点(3, 4)の間:  
距離: 5.00, 中点: (1.50, 2.00)  
  
半径5の円:  
面積: 78.54, 周囲: 31.42  
  
統計計算:  
配列: [34, 12, 67, 89, 23, 45, 78, 56, 90, 11]  
最小: 11, 最大: 90, 平均: 50.50, 中央値: 50.50  
  
成績情報:  
点数: [85, 92, 78, 65, 98, 55, 73, 88]  
平均: 79.25, 評価: B, 合格者数: 6

### 問題11-5: エラーハンドリング付き関数

安全性を考慮した以下の関数を作成してください。

1. **安全な基本操作**
   * 安全な除算関数 int safe\_divide(double a, double b, double \*result)
   * 安全な配列アクセス関数 int safe\_array\_get(int arr[], int size, int index, int \*value)
   * 安全な文字列コピー関数 int safe\_strcpy(char dest[], int dest\_size, const char src[])
2. **範囲チェック付き関数**
   * 範囲指定付きランダム数生成関数 int random\_range(int min, int max, int \*result)
   * 配列の範囲チェック付き設定関数 int safe\_array\_set(int arr[], int size, int index, int value)
3. **バリデーション関数**
   * 文字列が数値として有効かチェックする関数 int is\_valid\_number(const char str[])
   * メールアドレスの基本形式をチェックする関数 int is\_valid\_email(const char email[])

**期待される出力例:**

=== エラーハンドリング関数のテスト ===  
  
安全な除算:  
safe\_divide(10, 3) = 3.33 (成功)  
safe\_divide(10, 0) = エラー (ゼロ除算)  
  
安全な配列アクセス:  
safe\_array\_get(index=2) = 67 (成功)  
safe\_array\_get(index=15) = エラー (範囲外)  
  
安全な文字列コピー:  
safe\_strcpy("Hello") = 成功  
safe\_strcpy("VeryLongStringThatExceedsBufferSize") = エラー (バッファ不足)  
  
バリデーション:  
is\_valid\_number("123.45") = 1 (有効)  
is\_valid\_number("12.3.4") = 0 (無効)  
is\_valid\_email("user@example.com") = 1 (有効)  
is\_valid\_email("invalid.email") = 0 (無効)

### 問題11-6: 構造体を使った関数

以下の構造体を使った関数を作成してください。

typedef struct {  
 double x;  
 double y;  
} Point;  
  
typedef struct {  
 int id;  
 char name[50];  
 int age;  
 double score;  
} Student;  
  
typedef struct {  
 int year;  
 int month;  
 int day;  
} Date;

1. **座標操作**
   * 2点間の距離を計算する関数 double point\_distance(Point p1, Point p2)
   * 点を移動する関数 Point move\_point(Point p, double dx, double dy)
   * 複数の点の重心を計算する関数 Point calculate\_centroid(Point points[], int count)
2. **学生情報処理**
   * 学生情報を表示する関数 void print\_student(Student s)
   * 学生の成績を更新する関数 void update\_score(Student \*s, double new\_score)
   * 学生配列から最高得点者を見つける関数 Student find\_best\_student(Student students[], int count)
3. **日付操作**
   * 日付を表示する関数 void print\_date(Date d)
   * 日付の妥当性をチェックする関数 int is\_valid\_date(Date d)
   * 2つの日付の差を計算する関数 int date\_difference(Date d1, Date d2)

**期待される出力例:**

=== 構造体関数のテスト ===  
  
座標操作:  
点A(0, 0)と点B(3, 4)の距離: 5.00  
点(2, 3)を(1, -1)移動: (3, 2)  
3点の重心: (2.00, 3.00)  
  
学生情報処理:  
学生: ID=1001, 名前=田中太郎, 年齢=20, 成績=85.5  
成績更新後: 92.0  
最高得点者: 山田花子 (成績: 95.0)  
  
日付操作:  
日付: 2024/3/15  
is\_valid\_date(2024/2/30) = 0 (無効)  
is\_valid\_date(2024/3/15) = 1 (有効)  
2024/1/1 から 2024/3/15 までの日数: 74日

## チャレンジ問題

### 問題11-7: 再帰関数

以下の再帰関数を作成してください。

1. **数学的再帰**
   * 階乗を計算する再帰関数 long factorial\_recursive(int n)
   * フィボナッチ数列を計算する再帰関数 int fibonacci\_recursive(int n)
   * ユークリッドの互除法による最大公約数を求める再帰関数 int gcd\_recursive(int a, int b)
2. **文字列再帰**
   * 文字列が回文かどうか再帰的に判定する関数 int is\_palindrome\_recursive(const char str[], int start, int end)
   * 文字列を再帰的に逆順にする関数 void reverse\_string\_recursive(char str[], int start, int end)
3. **配列再帰**
   * 配列の合計を再帰的に計算する関数 int array\_sum\_recursive(int arr[], int size)
   * 配列の最大値を再帰的に見つける関数 int find\_max\_recursive(int arr[], int size)

**期待される出力例:**

=== 再帰関数のテスト ===  
  
数学的再帰:  
5! = 120  
fibonacci(10) = 55  
gcd(48, 18) = 6  
  
文字列再帰:  
"racecar" は回文: Yes  
"hello" は回文: No  
"Hello" の逆順: "olleH"  
  
配列再帰:  
配列 [1, 2, 3, 4, 5] の合計: 15  
配列 [34, 67, 12, 89, 23] の最大値: 89

### 問題11-8: 高度な文字列処理

以下の高度な文字列処理関数を作成してください。

1. **文字列解析**
   * 文字列をトークンに分割する関数 int tokenize(char str[], char tokens[][50], char delimiter)
   * 文字列内の括弧の対応をチェックする関数 int check\_brackets(const char str[])
   * 文字列から数値を抽出する関数 int extract\_numbers(const char str[], int numbers[], int max\_count)
2. **パターンマッチング**
   * 簡単なワイルドカード（\*、?）パターンマッチング関数 int wildcard\_match(const char str[], const char pattern[])
   * 文字列の置換関数 int string\_replace(char str[], const char old\_substr[], const char new\_substr[])
3. **文字列フォーマット**
   * 文字列を指定幅で中央揃えする関数 void center\_string(char result[], const char str[], int width)
   * CSV形式の文字列を解析する関数 int parse\_csv\_line(const char line[], char fields[][100], int max\_fields)

### 問題11-9: ソートアルゴリズム関数

以下のソートアルゴリズムを関数として実装してください。

1. **基本ソート**
   * バブルソート void bubble\_sort(int arr[], int size)
   * 選択ソート void selection\_sort(int arr[], int size)
   * 挿入ソート void insertion\_sort(int arr[], int size)
2. **高速ソート**
   * マージソート void merge\_sort(int arr[], int left, int right)
   * クイックソート void quick\_sort(int arr[], int left, int right)
3. **特殊ソート**
   * 文字列配列のソート void sort\_strings(char strings[][100], int count)
   * 構造体配列のソート（複数キー対応）void sort\_students(Student students[], int count, int sort\_by)

## 挑戦課題

### 課題9-10: 関数ポインター

以下の関数ポインターを使った課題に取り組んでください。

1. **計算機関数**
   * 四則演算を関数ポインターで切り替える計算機 double calculator(double a, double b, double (\*operation)(double, double))
   * 配列に対する処理を関数ポインターで指定する関数 void process\_array(int arr[], int size, void (\*processor)(int\*))
2. **ソート関数の汎用化**
   * 比較関数を引数に取る汎用ソート関数 void generic\_sort(void \*arr, int size, int elem\_size, int (\*compare)(const void\*, const void\*))
3. **コールバック関数**
   * イベント処理システム void register\_callback(int event\_type, void (\*callback)(int))

### 課題9-11: メモリ効率を考慮した関数

メモリ使用量を最適化した関数を作成してください。

1. **動的メモリ管理**
   * 動的配列を管理する関数群
   * 文字列プールを管理する関数群
   * メモリリークを検出する関数
2. **効率的なアルゴリズム**
   * インプレース（元の配列内で処理）ソート関数
   * メモリ使用量を抑えた文字列処理関数
   * キャッシュ効率を考慮した配列処理関数

### 課題9-12: 総合プロジェクト

複数の関数を組み合わせた総合的なプロジェクトを作成してください。

1. **学生管理システム**
   * 学生情報の登録、検索、更新、削除機能
   * 成績統計の計算機能
   * データのファイル保存・読み込み機能
2. **テキスト解析ツール**
   * ファイルの読み込みと解析
   * 単語頻度の統計
   * 文字列パターンの検索
3. **数値計算ライブラリ**
   * 行列演算機能
   * 統計計算機能
   * 数値積分・微分機能

## 提出要件

各課題について、以下を提出してください。

1. **ソースコード**
   * 関数の実装（.cファイル）
   * ヘッダファイル（.hファイル）
   * テスト用のmain関数
2. **ドキュメント**
   * 各関数の仕様説明
   * 使用例とサンプル出力
   * エラーケースの説明
3. **テストケース**
   * 正常ケースのテスト
   * 境界値のテスト
   * エラーケースのテスト

## 評価ポイント

* **正確性**: 仕様通りに動作するか
* **安全性**: エラーハンドリングが適切か
* **効率性**: アルゴリズムの計算量は適切か
* **可読性**: コードが理解しやすいか
* **再利用性**: 他のプログラムでも使えるか

## 難易度別の推奨学習順序

### 初学者向け（基本課題から開始）

1. 課題9-1: 基本的な関数作成
2. 課題9-2: 配列操作関数
3. 課題9-3: 文字列処理関数

### 中級者向け（中級課題に挑戦）

1. 課題9-4: 複数戻り値を持つ関数
2. 課題9-5: エラーハンドリング付き関数
3. 課題9-6: 構造体を使った関数

### 上級者向け（上級・挑戦課題）

1. 課題9-7: 再帰関数
2. 課題9-8: 高度な文字列処理
3. 課題9-9: ソートアルゴリズム関数
4. 課題9-10: 関数ポインター
5. 課題9-11: メモリ効率を考慮した関数
6. 課題9-12: 総合プロジェクト

## 学習のヒント

1. **段階的実装**
   * 最初は基本機能のみ実装
   * 動作確認後に機能を追加
   * テストを頻繁に実行
2. **エラーハンドリング**
   * NULLポインターのチェック
   * 配列の境界チェック
   * 無効な引数の処理
3. **関数設計の原則**
   * 単一責任の原則
   * 適切な関数名と引数名
   * const修飾子の活用
4. **テスト駆動開発**
   * 関数作成前にテストケースを考える
   * 境界値のテスト
   * エラーケースのテスト
5. **コードレビュー**
   * 他の人に読んでもらう
   * 改善点を見つける
   * ベストプラクティスを学ぶ

## 参考資料

* C言語関数リファレンス
* アルゴリズムとデータ構造の教科書
* セキュアプログラミングガイド
* 関数型プログラミングの考え方

## 実装例の構成

各課題の実装では以下の構成を推奨します。

// ヘッダーファイル（functions.h）  
// - 関数プロトタイプ  
// - 定数定義  
// - 構造体定義  
  
// 実装ファイル（functions.c）  
// - 関数の実装  
// - 静的関数（内部使用）  
// - エラーハンドリング  
  
// テストファイル（test.c）  
// - main関数  
// - 各関数のテスト  
// - 結果の検証

## 発展的な学習内容

1. **可変長引数**
   * va\_list、va\_start、va\_endの使用
   * printfライクな関数の作成
2. **関数ポインター**
   * コールバック関数の実装
   * 関数テーブルの作成
3. **マクロ関数**
   * 関数ライクマクロの作成
   * 条件付きコンパイル
4. **インライン関数**
   * パフォーマンスの最適化
   * 適切な使用場面

このチュートリアルを通じて、C言語の関数について体系的に学習し、実践的なプログラミングスキルを身に付けてください。

# 演習問題を解き終えたら

## 理解度の確認

応用編の演習問題を通じて、以下の点を確認してください：

### ポインタの理解

* ポインタとアドレスの関係を説明できる
* ポインタ演算を正しく使える
* 配列とポインタの関係を理解している

### 文字列処理

* 文字配列と文字列の違いを理解している
* 標準ライブラリ関数を適切に使える
* 独自の文字列処理関数を実装できる

### 関数設計

* 適切な引数と戻り値を設計できる
* ポインタ渡しと値渡しを使い分けられる
* 再帰関数の動作を理解している

## 次のステップ

応用編を修了したら、上級編に進んでください。上級編では、より高度なデータ構造とシステムプログラミングの技術を学びます。

## デバッグのヒント

応用編では、特に以下のエラーに注意してください：

1. **セグメンテーションフォルト**
   * NULLポインタへのアクセス
   * 配列の範囲外アクセス
   * 解放済みメモリへのアクセス
2. **メモリリーク**
   * mallocしたメモリのfree忘れ
   * 適切なメモリ管理
3. **文字列処理のバグ**
   * NULL終端の忘れ
   * バッファオーバーフロー

これらのエラーは、printfデバッグやデバッガを使って丁寧に調査してください。