# 第9章 関数 - 演習問題

# 演習問題

## 基本課題

### 課題9-1: 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)

### 課題9-2: 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)

### 課題9-3: 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[])

## 中級課題

### 課題9-4: 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)

### 課題9-5: 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[])

### 課題9-6: 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)

## 上級課題

### 課題9-7: 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)

### 課題9-8: 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)

### 課題9-9: 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。

# 解答例

この章では、関数の基本から応用まで、様々な演習問題の解答例を提供しています。

## 解答ファイル一覧

### 演習9-1: 基本的な関数作成

素数判定や簡単な計算を行う基本的な関数群の実装例です。

* <ex9_1_prime_functions.c> - C90準拠版
* <ex9_1_prime_functions_c99.c> - C99準拠版（bool型、//コメント使用）

**学習ポイント:** - 関数の基本的な定義と呼び出し - 引数と戻り値の扱い方 - ループと条件分岐の組み合わせ

### 演習9-2: 配列操作関数

配列に対する各種操作（合計、平均、ソート等）を行う関数群の実装例です。

* <ex9_2_array_operations.c> - C90準拠版
* <ex9_2_array_operations_c99.c> - C99準拠版（可変長配列、inline関数使用）

**学習ポイント:** - 配列を引数として受け取る方法 - ポインターを使った配列操作 - 動的メモリ割り当て（calloc/free）

### 演習9-3: 文字列処理関数

文字列の操作（コピー、連結、変換等）を行う関数群の実装例です。

* <ex9_3_string_processing.c> - C90準拠版
* <ex9_3_string_processing_c99.c> - C99準拠版（bool型、inline関数使用）

**学習ポイント:** - 文字配列としての文字列操作 - 標準ライブラリ関数の再実装 - 文字列の解析と変換

### 演習9-4: 複数戻り値を持つ関数

ポインター引数を使って複数の値を返す関数群の実装例です。

* <ex9_4_multiple_returns.c> - C90準拠版
* <ex9_4_multiple_returns_c99.c> - C99準拠版（構造体戻り値、複合リテラル使用）

**学習ポイント:** - ポインター引数による複数値の返却 - 構造体を使った複数値のまとめ方 - 数学的計算と座標変換

### 演習9-5: エラーハンドリング付き関数

エラー処理を含む堅牢な関数群の実装例です。

* <ex9_5_error_handling.c> - C90準拠版
* <ex9_5_error_handling_c99.c> - C99準拠版（enum型、可変長引数マクロ使用）

**学習ポイント:** - エラーコードの設計と管理 - 安全な数値計算（オーバーフロー対策） - 境界チェックとNULLポインターチェック

### 演習9-6: 構造体を使った関数

構造体を引数や戻り値として使用する関数群の実装例です。

* <ex9_6_struct_functions.c> - C90準拠版
* <ex9_6_struct_functions_c99.c> - C99準拠版（designated initializer、複合リテラル使用）

**学習ポイント:** - 構造体の値渡しと参照渡し - 構造体を返す関数の実装 - 複雑なデータ構造の操作

## コンパイルと実行

各解答例は以下のようにコンパイルできます：

# C90版のコンパイル例  
gcc -Wall -Wextra -pedantic -std=c90 ex9\_1\_prime\_functions.c -o ex9\_1\_prime  
  
# C99版のコンパイル例（数学関数を使う場合は-lmを追加）  
gcc -Wall -Wextra -pedantic -std=c99 ex9\_4\_multiple\_returns\_c99.c -lm -o ex9\_4\_c99  
  
# 実行  
./ex9\_1\_prime

## 学習のポイント

### 1. C90とC99の違い

各演習にはC90版とC99版の両方を用意しています。主な違いは：

* **変数宣言**: C90は関数の先頭、C99は使用箇所で宣言可能
* **コメント**: C90は/\* \*/のみ、C99は//も使用可能
* **bool型**: C90は整数で代用、C99はstdbool.hのbool型
* **その他**: C99では可変長配列、複合リテラル、designated initializerなど

### 2. 関数設計の原則

* **単一責任の原則**: 1つの関数は1つの機能に集中
* **適切な名前付け**: 関数名から機能が推測できる
* **エラー処理**: 異常系を考慮した堅牢な実装
* **const修飾子**: 変更しない引数にはconstを付ける

### 3. テストの重要性

各解答例にはテスト用のmain関数が含まれています：

* 正常系のテスト
* 異常系のテスト（エラーケース）
* 境界値のテスト
* 実行結果の表示

### 4. メモリ管理

動的メモリを使用する場合：

* malloc/callocで確保
* 使用後は必ずfreeで解放
* NULLチェックを忘れない
* メモリリークに注意

### 5. 移植性の考慮

* 標準ライブラリのみを使用
* プラットフォーム依存のコードを避ける
* 適切な型の使用（size\_t、int64\_tなど）

## 発展的な学習

これらの基本的な実装を理解したら、以下に挑戦してみましょう：

1. **最適化**: より効率的なアルゴリズムの実装
2. **拡張**: 新しい機能の追加
3. **汎用化**: より汎用的な関数の設計
4. **エラー処理の改善**: より詳細なエラー情報の提供

## 注意事項

* 実装例はあくまで一例です。他の実装方法も検討してみましょう
* 実際のプロジェクトでは、標準ライブラリの使用を推奨します
* セキュリティを考慮した実装を心がけましょう ## ex9\_1\_prime\_functions.c

/\*  
 \* 演習解答例: 素数判定関数とその活用  
 \*  
 \* このプログラムは、素数判定を行う関数を作成し、  
 \* それを使って様々な処理を行います。  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
int is\_prime(int n);  
void print\_primes\_in\_range(int start, int end);  
int count\_primes\_in\_range(int start, int end);  
int nth\_prime(int n);  
  
/\* 素数判定を行う関数 \*/  
int is\_prime(int n)  
{  
 int i;  
  
 /\* 1以下は素数ではない \*/  
 if (n <= 1)  
 {  
 return 0;  
 }  
  
 /\* 2は素数 \*/  
 if (n == 2)  
 {  
 return 1;  
 }  
  
 /\* 偶数は2以外素数ではない \*/  
 if (n % 2 == 0)  
 {  
 return 0;  
 }  
  
 /\* 3からsqrt(n)まで奇数のみをチェック \*/  
 for (i = 3; i \* i <= n; i += 2)  
 {  
 if (n % i == 0)  
 {  
 return 0; /\* 約数が見つかった \*/  
 }  
 }  
  
 return 1; /\* 素数 \*/  
}  
  
/\* 指定した範囲の素数を表示する関数 \*/  
void print\_primes\_in\_range(int start, int end)  
{  
 int i;  
 int count = 0;  
  
 printf("%dから%dまでの素数:\n", start, end);  
  
 for (i = start; i <= end; i++)  
 {  
 if (is\_prime(i))  
 {  
 printf("%d ", i);  
 count++;  
  
 /\* 10個ごとに改行 \*/  
 if (count % 10 == 0)  
 {  
 printf("\n");  
 }  
 }  
 }  
  
 if (count % 10 != 0)  
 {  
 printf("\n");  
 }  
  
 printf("見つかった素数の個数: %d\n", count);  
}  
  
/\* 指定した範囲の素数の個数を数える関数 \*/  
int count\_primes\_in\_range(int start, int end)  
{  
 int i;  
 int count = 0;  
  
 for (i = start; i <= end; i++)  
 {  
 if (is\_prime(i))  
 {  
 count++;  
 }  
 }  
  
 return count;  
}  
  
/\* n番目の素数を求める関数 \*/  
int nth\_prime(int n)  
{  
 int count = 0;  
 int num = 2;  
  
 if (n <= 0)  
 {  
 return -1; /\* 無効な入力 \*/  
 }  
  
 while (count < n)  
 {  
 if (is\_prime(num))  
 {  
 count++;  
 if (count == n)  
 {  
 return num;  
 }  
 }  
 num++;  
 }  
  
 return -1; /\* ここには到達しないはず \*/  
}  
  
int main(void)  
{  
 printf("=== 素数判定プログラム（デモ版）===\n\n");  
  
 /\* デモ1: 単一の数の素数判定 \*/  
 printf("1. 単一の数の素数判定\n");  
 int test\_numbers[] = {2, 3, 4, 17, 18, 97, 100, 101};  
 int num\_tests = sizeof(test\_numbers) / sizeof(test\_numbers[0]);  
 int i;  
   
 for (i = 0; i < num\_tests; i++)  
 {  
 int number = test\_numbers[i];  
 printf(" %d -> ", number);  
 if (is\_prime(number))  
 {  
 printf("素数です\n");  
 }  
 else  
 {  
 printf("素数ではありません\n");  
 }  
 }  
 printf("\n");  
  
 /\* デモ2: 範囲内の素数を表示 \*/  
 printf("2. 範囲内の素数を表示\n");  
 int demo\_ranges[][2] = {{1, 20}, {30, 50}, {90, 110}};  
 int num\_ranges = sizeof(demo\_ranges) / sizeof(demo\_ranges[0]);  
   
 for (i = 0; i < num\_ranges; i++)  
 {  
 int start = demo\_ranges[i][0];  
 int end = demo\_ranges[i][1];  
 printf(" 範囲 %d から %d の素数:\n ", start, end);  
 print\_primes\_in\_range(start, end);  
 printf("\n");  
 }  
  
 /\* デモ3: 範囲内の素数の個数をカウント \*/  
 printf("3. 範囲内の素数の個数をカウント\n");  
 for (i = 0; i < num\_ranges; i++)  
 {  
 int start = demo\_ranges[i][0];  
 int end = demo\_ranges[i][1];  
 int count = count\_primes\_in\_range(start, end);  
 printf(" 範囲 %d から %d: %d個\n", start, end, count);  
 }  
 printf("\n");  
  
 /\* デモ4: n番目の素数を求める \*/  
 printf("4. n番目の素数を求める\n");  
 int nth\_tests[] = {1, 5, 10, 15, 20, 25};  
 int nth\_count = sizeof(nth\_tests) / sizeof(nth\_tests[0]);  
   
 for (i = 0; i < nth\_count; i++)  
 {  
 int n = nth\_tests[i];  
 printf(" %d番目の素数: %d\n", n, nth\_prime(n));  
 }  
 printf("\n");  
  
 /\* 追加デモ: 最初の30個の素数を表示 \*/  
 printf("5. 最初の30個の素数\n ");  
 for (i = 1; i <= 30; i++)  
 {  
 printf("%d ", nth\_prime(i));  
 if (i % 10 == 0) printf("\n ");  
 }  
 printf("\n\n");  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}

## ex9\_2\_array\_operations.c

/\*  
 \* 演習9-2: 配列操作関数 - 解答例  
 \*   
 \* 配列に対する各種操作を行う関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <stdlib.h> /\* calloc, free用 \*/  
  
/\* 関数プロトタイプ \*/  
/\* 基本操作 \*/  
int array\_sum(int arr[], int size);  
double array\_average(int arr[], int size);  
int find\_max\_index(int arr[], int size);  
  
/\* 検索・ソート \*/  
int linear\_search(int arr[], int size, int target);  
void sort\_array(int arr[], int size);  
void reverse\_array(int arr[], int size);  
  
/\* 配列統計 \*/  
void find\_min\_max(int arr[], int size, int \*min, int \*max);  
int count\_duplicates(int arr[], int size);  
  
/\* ヘルパー関数 \*/  
void print\_array(int arr[], int size);  
void swap(int \*a, int \*b);  
  
/\* 配列の要素の合計を計算する関数 \*/  
int array\_sum(int arr[], int size)  
{  
 int sum = 0;  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 sum += arr[i];  
 }  
   
 return sum;  
}  
  
/\* 配列の平均値を計算する関数 \*/  
double array\_average(int arr[], int size)  
{  
 if (size == 0)  
 {  
 return 0.0;  
 }  
   
 return (double)array\_sum(arr, size) / 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)  
{  
 int max\_index = 0;  
 int i;  
   
 if (size <= 0)  
 {  
 return -1; /\* エラー \*/  
 }  
   
 for (i = 1; i < size; i++)  
 {  
 if (arr[i] > arr[max\_index])  
 {  
 max\_index = i;  
 }  
 }  
   
 return max\_index;  
}  
  
/\* 線形検索関数（見つかったらインデックス、見つからなければ-1を返す） \*/  
int linear\_search(int arr[], int size, int target)  
{  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 if (arr[i] == target)  
 {  
 return i;  
 }  
 }  
   
 return -1; /\* 見つからなかった \*/  
}  
  
/\* 配列を昇順にソートする関数（バブルソート） \*/  
void sort\_array(int arr[], int size)  
{  
 int i, j;  
   
 for (i = 0; i < size - 1; i++)  
 {  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 swap(&arr[j], &arr[j + 1]);  
 }  
 }  
 }  
}  
  
/\* 配列の要素を逆順にする関数 \*/  
void reverse\_array(int arr[], int size)  
{  
 int i;  
   
 for (i = 0; i < size / 2; i++)  
 {  
 swap(&arr[i], &arr[size - 1 - i]);  
 }  
}  
  
/\* 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)  
{  
 int i;  
   
 if (size <= 0 || min == NULL || max == NULL)  
 {  
 return;  
 }  
   
 \*min = arr[0];  
 \*max = arr[0];  
   
 for (i = 1; i < size; i++)  
 {  
 if (arr[i] < \*min)  
 {  
 \*min = arr[i];  
 }  
 if (arr[i] > \*max)  
 {  
 \*max = arr[i];  
 }  
 }  
}  
  
/\* 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)  
{  
 int count = 0;  
 int i, j;  
 int \*counted;  
   
 if (size <= 1)  
 {  
 return 0;  
 }  
   
 /\* カウント済みフラグ配列を作成 \*/  
 counted = (int\*)calloc(size, sizeof(int));  
 if (counted == NULL)  
 {  
 return -1; /\* メモリ割り当てエラー \*/  
 }  
   
 for (i = 0; i < size - 1; i++)  
 {  
 if (counted[i])  
 {  
 continue; /\* 既にカウント済み \*/  
 }  
   
 int dup\_count = 0;  
 for (j = i + 1; j < size; j++)  
 {  
 if (arr[i] == arr[j])  
 {  
 dup\_count++;  
 counted[j] = 1;  
 }  
 }  
   
 if (dup\_count > 0)  
 {  
 count += dup\_count;  
 }  
 }  
   
 free(counted);  
 return count;  
}  
  
/\* 配列を表示するヘルパー関数 \*/  
void print\_array(int arr[], int size)  
{  
 int i;  
   
 printf("[");  
 for (i = 0; i < size; i++)  
 {  
 printf("%d", arr[i]);  
 if (i < size - 1)  
 {  
 printf(", ");  
 }  
 }  
 printf("]\n");  
}  
  
/\* 2つの要素を交換するヘルパー関数 \*/  
void swap(int \*a, int \*b)  
{  
 int temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 int test\_array[] = {5, 2, 8, 2, 9, 1, 5, 5, 3, 7};  
 int size = sizeof(test\_array) / sizeof(test\_array[0]);  
 int min, max;  
 int search\_target;  
 int result;  
   
 printf("=== 配列操作関数のテスト ===\n\n");  
   
 /\* 元の配列を表示 \*/  
 printf("元の配列: ");  
 print\_array(test\_array, size);  
 printf("\n");  
   
 /\* 基本操作のテスト \*/  
 printf("=== 基本操作 ===\n");  
 printf("合計: %d\n", array\_sum(test\_array, size));  
 printf("平均: %.2f\n", array\_average(test\_array, size));  
 printf("最大値のインデックス: %d (値: %d)\n",   
 find\_max\_index(test\_array, size),   
 test\_array[find\_max\_index(test\_array, size)]);  
 printf("\n");  
   
 /\* 検索のテスト \*/  
 printf("=== 線形検索 ===\n");  
 search\_target = 8;  
 result = linear\_search(test\_array, size, search\_target);  
 if (result != -1)  
 {  
 printf("%d はインデックス %d で見つかりました\n", search\_target, result);  
 }  
 else  
 {  
 printf("%d は見つかりませんでした\n", search\_target);  
 }  
   
 search\_target = 10;  
 result = linear\_search(test\_array, size, search\_target);  
 if (result != -1)  
 {  
 printf("%d はインデックス %d で見つかりました\n", search\_target, result);  
 }  
 else  
 {  
 printf("%d は見つかりませんでした\n", search\_target);  
 }  
 printf("\n");  
   
 /\* 統計情報のテスト \*/  
 printf("=== 統計情報 ===\n");  
 find\_min\_max(test\_array, size, &min, &max);  
 printf("最小値: %d, 最大値: %d\n", min, max);  
 printf("重複する要素の数: %d\n", count\_duplicates(test\_array, size));  
 printf("\n");  
   
 /\* 配列のコピーを作成（ソート・逆順用） \*/  
 int work\_array[10];  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 work\_array[i] = test\_array[i];  
 }  
   
 /\* ソートのテスト \*/  
 printf("=== ソート ===\n");  
 printf("ソート前: ");  
 print\_array(work\_array, size);  
 sort\_array(work\_array, size);  
 printf("ソート後: ");  
 print\_array(work\_array, size);  
 printf("\n");  
   
 /\* 逆順のテスト \*/  
 printf("=== 逆順 ===\n");  
 for (i = 0; i < size; i++)  
 {  
 work\_array[i] = test\_array[i];  
 }  
 printf("逆順前: ");  
 print\_array(work\_array, size);  
 reverse\_array(work\_array, size);  
 printf("逆順後: ");  
 print\_array(work\_array, size);  
 printf("\n");  
   
 /\* エッジケースのテスト \*/  
 printf("=== エッジケースのテスト ===\n");  
 int empty\_array[1];  
 int single\_element[] = {42};  
   
 printf("空配列の平均: %.2f\n", array\_average(empty\_array, 0));  
 printf("単一要素配列の最大値インデックス: %d\n",   
 find\_max\_index(single\_element, 1));  
 printf("単一要素配列の重複数: %d\n",   
 count\_duplicates(single\_element, 1));  
   
 return 0;  
}```  
  
## ex9\_3\_string\_processing.c  
  
```c  
/\*  
 \* 演習9-3: 文字列処理関数 - 解答例  
 \*   
 \* 文字列に対する各種処理を行う関数群を実装します。  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
/\* 基本処理 \*/  
void my\_strcpy(char dest[], const char src[]);  
void my\_strcat(char dest[], const char src[]);  
int my\_strcmp(const char str1[], const char str2[]);  
  
/\* 変換処理 \*/  
void to\_uppercase(char str[]);  
void to\_lowercase(char str[]);  
void reverse\_string(char str[]);  
  
/\* 解析処理 \*/  
int count\_words(const char str[]);  
int is\_palindrome(const char str[]);  
int count\_vowels(const char str[]);  
  
/\* ヘルパー関数 \*/  
int my\_strlen(const char str[]);  
int is\_space(char c);  
int is\_alpha(char c);  
int is\_vowel(char c);  
  
/\* 文字列の長さを計算する関数 \*/  
int my\_strlen(const char str[])  
{  
 int len = 0;  
   
 while (str[len] != '\0')  
 {  
 len++;  
 }  
   
 return len;  
}  
  
/\* 文字列をコピーする関数 \*/  
void my\_strcpy(char dest[], const char src[])  
{  
 int i = 0;  
   
 while (src[i] != '\0')  
 {  
 dest[i] = src[i];  
 i++;  
 }  
 dest[i] = '\0';  
}  
  
/\* 文字列を連結する関数 \*/  
void my\_strcat(char dest[], const char src[])  
{  
 int dest\_len = my\_strlen(dest);  
 int i = 0;  
   
 while (src[i] != '\0')  
 {  
 dest[dest\_len + i] = src[i];  
 i++;  
 }  
 dest[dest\_len + i] = '\0';  
}  
  
/\* 文字列を比較する関数 \*/  
int my\_strcmp(const char str1[], const char str2[])  
{  
 int i = 0;  
   
 while (str1[i] != '\0' && str2[i] != '\0')  
 {  
 if (str1[i] < str2[i])  
 {  
 return -1;  
 }  
 else if (str1[i] > str2[i])  
 {  
 return 1;  
 }  
 i++;  
 }  
   
 /\* 片方の文字列が終了した場合 \*/  
 if (str1[i] == '\0' && str2[i] == '\0')  
 {  
 return 0; /\* 同じ \*/  
 }  
 else if (str1[i] == '\0')  
 {  
 return -1; /\* str1が短い \*/  
 }  
 else  
 {  
 return 1; /\* str2が短い \*/  
 }  
}  
  
/\* 文字列を大文字に変換する関数 \*/  
void to\_uppercase(char str[])  
{  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (str[i] >= 'a' && str[i] <= 'z')  
 {  
 str[i] = str[i] - 'a' + 'A';  
 }  
 i++;  
 }  
}  
  
/\* 文字列を小文字に変換する関数 \*/  
void to\_lowercase(char str[])  
{  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (str[i] >= 'A' && str[i] <= 'Z')  
 {  
 str[i] = str[i] - 'A' + 'a';  
 }  
 i++;  
 }  
}  
  
/\* 文字列を逆順にする関数 \*/  
void reverse\_string(char str[])  
{  
 int len = my\_strlen(str);  
 int i;  
 char temp;  
   
 for (i = 0; i < len / 2; i++)  
 {  
 temp = str[i];  
 str[i] = str[len - 1 - i];  
 str[len - 1 - i] = temp;  
 }  
}  
  
/\* 文字列内の単語数を数える関数 \*/  
int count\_words(const char str[])  
{  
 int count = 0;  
 int in\_word = 0;  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (is\_space(str[i]))  
 {  
 in\_word = 0;  
 }  
 else  
 {  
 if (!in\_word)  
 {  
 count++;  
 in\_word = 1;  
 }  
 }  
 i++;  
 }  
   
 return count;  
}  
  
/\* 文字列が回文かどうかを判定する関数 \*/  
int is\_palindrome(const char str[])  
{  
 int len = my\_strlen(str);  
 int i, j;  
   
 /\* 空白を無視してチェック \*/  
 i = 0;  
 j = len - 1;  
   
 while (i < j)  
 {  
 /\* 左側の空白をスキップ \*/  
 while (i < j && is\_space(str[i]))  
 {  
 i++;  
 }  
   
 /\* 右側の空白をスキップ \*/  
 while (i < j && is\_space(str[j]))  
 {  
 j--;  
 }  
   
 /\* 大文字小文字を無視して比較 \*/  
 char left = str[i];  
 char right = str[j];  
   
 if (left >= 'A' && left <= 'Z')  
 {  
 left = left - 'A' + 'a';  
 }  
 if (right >= 'A' && right <= 'Z')  
 {  
 right = right - 'A' + 'a';  
 }  
   
 if (left != right)  
 {  
 return 0; /\* 回文ではない \*/  
 }  
   
 i++;  
 j--;  
 }  
   
 return 1; /\* 回文 \*/  
}  
  
/\* 文字列内の母音の数を数える関数 \*/  
int count\_vowels(const char str[])  
{  
 int count = 0;  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (is\_vowel(str[i]))  
 {  
 count++;  
 }  
 i++;  
 }  
   
 return count;  
}  
  
/\* 文字が空白かどうかを判定するヘルパー関数 \*/  
int is\_space(char c)  
{  
 return c == ' ' || c == '\t' || c == '\n';  
}  
  
/\* 文字がアルファベットかどうかを判定するヘルパー関数 \*/  
int is\_alpha(char c)  
{  
 return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');  
}  
  
/\* 文字が母音かどうかを判定するヘルパー関数 \*/  
int is\_vowel(char c)  
{  
 char lower = c;  
   
 if (c >= 'A' && c <= 'Z')  
 {  
 lower = c - 'A' + 'a';  
 }  
   
 return lower == 'a' || lower == 'e' || lower == 'i' ||   
 lower == 'o' || lower == 'u';  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 char str1[100] = "Hello, World!";  
 char str2[100] = "C Programming";  
 char str3[100] = "";  
 char str4[100] = "A man a plan a canal Panama";  
 char str5[100] = "race car";  
 char buffer[200];  
   
 printf("=== 文字列処理関数のテスト ===\n\n");  
   
 /\* 基本処理のテスト \*/  
 printf("=== 基本処理 ===\n");  
 printf("str1: \"%s\"\n", str1);  
 printf("str2: \"%s\"\n", str2);  
 printf("strlen(str1): %d\n", my\_strlen(str1));  
 printf("strlen(str2): %d\n", my\_strlen(str2));  
   
 /\* strcpyのテスト \*/  
 my\_strcpy(str3, str1);  
 printf("\nstrcpy(str3, str1)後: \"%s\"\n", str3);  
   
 /\* strcatのテスト \*/  
 my\_strcpy(buffer, str1);  
 my\_strcat(buffer, " ");  
 my\_strcat(buffer, str2);  
 printf("strcat結果: \"%s\"\n", buffer);  
   
 /\* strcmpのテスト \*/  
 printf("\nstrcmp(str1, str2): %d\n", my\_strcmp(str1, str2));  
 printf("strcmp(str1, str1): %d\n", my\_strcmp(str1, str1));  
 printf("strcmp(\"ABC\", \"ABD\"): %d\n", my\_strcmp("ABC", "ABD"));  
 printf("\n");  
   
 /\* 変換処理のテスト \*/  
 printf("=== 変換処理 ===\n");  
 char test\_str[100];  
   
 /\* 大文字変換 \*/  
 my\_strcpy(test\_str, "Hello World 123!");  
 printf("元の文字列: \"%s\"\n", test\_str);  
 to\_uppercase(test\_str);  
 printf("大文字変換: \"%s\"\n", test\_str);  
   
 /\* 小文字変換 \*/  
 to\_lowercase(test\_str);  
 printf("小文字変換: \"%s\"\n", test\_str);  
   
 /\* 逆順 \*/  
 my\_strcpy(test\_str, "abcdef");  
 printf("\n元の文字列: \"%s\"\n", test\_str);  
 reverse\_string(test\_str);  
 printf("逆順: \"%s\"\n", test\_str);  
 printf("\n");  
   
 /\* 解析処理のテスト \*/  
 printf("=== 解析処理 ===\n");  
   
 /\* 単語数カウント \*/  
 char sentence1[] = "The quick brown fox jumps over the lazy dog";  
 char sentence2[] = " One two three ";  
 char sentence3[] = "";  
   
 printf("\"%s\"\n単語数: %d\n\n", sentence1, count\_words(sentence1));  
 printf("\"%s\"\n単語数: %d\n\n", sentence2, count\_words(sentence2));  
 printf("\"%s\"\n単語数: %d\n\n", sentence3, count\_words(sentence3));  
   
 /\* 回文判定 \*/  
 printf("回文判定:\n");  
 printf("\"%s\": %s\n", str4, is\_palindrome(str4) ? "回文" : "回文ではない");  
 printf("\"%s\": %s\n", str5, is\_palindrome(str5) ? "回文" : "回文ではない");  
 printf("\"%s\": %s\n", str1, is\_palindrome(str1) ? "回文" : "回文ではない");  
   
 /\* 母音カウント \*/  
 printf("\n母音の数:\n");  
 printf("\"%s\": %d個\n", str1, count\_vowels(str1));  
 printf("\"%s\": %d個\n", str2, count\_vowels(str2));  
 printf("\"%s\": %d個\n", "AEIOU aeiou", count\_vowels("AEIOU aeiou"));  
   
 return 0;  
}```  
  
## ex9\_4\_multiple\_returns.c  
  
```c  
/\*  
 \* 演習9-4: 複数戻り値を持つ関数 - 解答例  
 \*   
 \* ポインター引数を使って複数の値を返す関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <math.h>  
  
/\* 関数プロトタイプ \*/  
/\* 時間変換 \*/  
void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds);  
void time\_add(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*result\_h, int \*result\_m, int \*result\_s);  
void time\_diff(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*diff\_h, int \*diff\_m, int \*diff\_s);  
  
/\* 座標計算 \*/  
void polar\_to\_cartesian(double r, double theta, double \*x, double \*y);  
void cartesian\_to\_polar(double x, double y, double \*r, double \*theta);  
void midpoint(double x1, double y1, double x2, double y2, double \*mid\_x, double \*mid\_y);  
  
/\* 統計計算 \*/  
void basic\_statistics(double data[], int size, double \*mean, double \*variance, double \*std\_dev);  
void min\_max\_range(double data[], int size, double \*min, double \*max, double \*range);  
void percentiles(double sorted\_data[], int size, double \*q1, double \*median, double \*q3);  
  
/\* ヘルパー関数 \*/  
void sort\_doubles(double arr[], int size);  
void swap\_double(double \*a, double \*b);  
  
/\* 秒を時・分・秒に変換する関数 \*/  
void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds)  
{  
 if (hours == NULL || minutes == NULL || seconds == NULL)  
 {  
 return;  
 }  
   
 \*hours = total\_seconds / 3600;  
 total\_seconds %= 3600;  
 \*minutes = total\_seconds / 60;  
 \*seconds = total\_seconds % 60;  
}  
  
/\* 2つの時刻を加算する関数 \*/  
void time\_add(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*result\_h, int \*result\_m, int \*result\_s)  
{  
 if (result\_h == NULL || result\_m == NULL || result\_s == NULL)  
 {  
 return;  
 }  
   
 int total\_seconds = (h1 \* 3600 + m1 \* 60 + s1) + (h2 \* 3600 + m2 \* 60 + s2);  
 seconds\_to\_hms(total\_seconds, result\_h, result\_m, result\_s);  
}  
  
/\* 2つの時刻の差を計算する関数 \*/  
void time\_diff(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*diff\_h, int \*diff\_m, int \*diff\_s)  
{  
 if (diff\_h == NULL || diff\_m == NULL || diff\_s == NULL)  
 {  
 return;  
 }  
   
 int seconds1 = h1 \* 3600 + m1 \* 60 + s1;  
 int seconds2 = h2 \* 3600 + m2 \* 60 + s2;  
 int diff\_seconds = seconds1 - seconds2;  
   
 /\* 負の場合は正に変換 \*/  
 if (diff\_seconds < 0)  
 {  
 diff\_seconds = -diff\_seconds;  
 }  
   
 seconds\_to\_hms(diff\_seconds, diff\_h, diff\_m, diff\_s);  
}  
  
/\* 極座標を直交座標に変換する関数 \*/  
void polar\_to\_cartesian(double r, double theta, double \*x, double \*y)  
{  
 if (x == NULL || y == NULL)  
 {  
 return;  
 }  
   
 \*x = r \* cos(theta);  
 \*y = r \* sin(theta);  
}  
  
/\* 直交座標を極座標に変換する関数 \*/  
void cartesian\_to\_polar(double x, double y, double \*r, double \*theta)  
{  
 if (r == NULL || theta == NULL)  
 {  
 return;  
 }  
   
 \*r = sqrt(x \* x + y \* y);  
 \*theta = atan2(y, x);  
}  
  
/\* 2点の中点を求める関数 \*/  
void midpoint(double x1, double y1, double x2, double y2, double \*mid\_x, double \*mid\_y)  
{  
 if (mid\_x == NULL || mid\_y == NULL)  
 {  
 return;  
 }  
   
 \*mid\_x = (x1 + x2) / 2.0;  
 \*mid\_y = (y1 + y2) / 2.0;  
}  
  
/\* 基本統計量を計算する関数 \*/  
void basic\_statistics(double data[], int size, double \*mean, double \*variance, double \*std\_dev)  
{  
 if (mean == NULL || variance == NULL || std\_dev == NULL || size <= 0)  
 {  
 return;  
 }  
   
 /\* 平均値を計算 \*/  
 double sum = 0.0;  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 sum += data[i];  
 }  
 \*mean = sum / size;  
   
 /\* 分散を計算 \*/  
 double sum\_sq\_diff = 0.0;  
 for (i = 0; i < size; i++)  
 {  
 double diff = data[i] - \*mean;  
 sum\_sq\_diff += diff \* diff;  
 }  
 \*variance = sum\_sq\_diff / size;  
   
 /\* 標準偏差を計算 \*/  
 \*std\_dev = sqrt(\*variance);  
}  
  
/\* 最小値、最大値、範囲を求める関数 \*/  
void min\_max\_range(double data[], int size, double \*min, double \*max, double \*range)  
{  
 if (min == NULL || max == NULL || range == NULL || size <= 0)  
 {  
 return;  
 }  
   
 \*min = data[0];  
 \*max = data[0];  
   
 int i;  
 for (i = 1; i < size; i++)  
 {  
 if (data[i] < \*min)  
 {  
 \*min = data[i];  
 }  
 if (data[i] > \*max)  
 {  
 \*max = data[i];  
 }  
 }  
   
 \*range = \*max - \*min;  
}  
  
/\* パーセンタイル（四分位数）を計算する関数 \*/  
void percentiles(double sorted\_data[], int size, double \*q1, double \*median, double \*q3)  
{  
 if (q1 == NULL || median == NULL || q3 == NULL || size <= 0)  
 {  
 return;  
 }  
   
 /\* 中央値（第2四分位数） \*/  
 if (size % 2 == 0)  
 {  
 \*median = (sorted\_data[size/2 - 1] + sorted\_data[size/2]) / 2.0;  
 }  
 else  
 {  
 \*median = sorted\_data[size/2];  
 }  
   
 /\* 第1四分位数 \*/  
 int q1\_pos = size / 4;  
 if (size % 4 == 0)  
 {  
 \*q1 = (sorted\_data[q1\_pos - 1] + sorted\_data[q1\_pos]) / 2.0;  
 }  
 else  
 {  
 \*q1 = sorted\_data[q1\_pos];  
 }  
   
 /\* 第3四分位数 \*/  
 int q3\_pos = (3 \* size) / 4;  
 if ((3 \* size) % 4 == 0)  
 {  
 \*q3 = (sorted\_data[q3\_pos - 1] + sorted\_data[q3\_pos]) / 2.0;  
 }  
 else  
 {  
 \*q3 = sorted\_data[q3\_pos];  
 }  
}  
  
/\* 配列をソートするヘルパー関数 \*/  
void sort\_doubles(double arr[], int size)  
{  
 int i, j;  
 for (i = 0; i < size - 1; i++)  
 {  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 swap\_double(&arr[j], &arr[j + 1]);  
 }  
 }  
 }  
}  
  
/\* 2つの要素を交換するヘルパー関数 \*/  
void swap\_double(double \*a, double \*b)  
{  
 double temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 printf("=== 複数戻り値を持つ関数のテスト ===\n\n");  
   
 /\* 時間変換のテスト \*/  
 printf("=== 時間変換 ===\n");  
 int hours, minutes, seconds;  
   
 seconds\_to\_hms(7265, &hours, &minutes, &seconds);  
 printf("7265秒 = %d時間 %d分 %d秒\n", hours, minutes, seconds);  
   
 int h1 = 2, m1 = 45, s1 = 30;  
 int h2 = 1, m2 = 50, s2 = 40;  
 int result\_h, result\_m, result\_s;  
   
 time\_add(h1, m1, s1, h2, m2, s2, &result\_h, &result\_m, &result\_s);  
 printf("%d:%02d:%02d + %d:%02d:%02d = %d:%02d:%02d\n",   
 h1, m1, s1, h2, m2, s2, result\_h, result\_m, result\_s);  
   
 time\_diff(h1, m1, s1, h2, m2, s2, &result\_h, &result\_m, &result\_s);  
 printf("%d:%02d:%02d - %d:%02d:%02d = %d:%02d:%02d\n\n",   
 h1, m1, s1, h2, m2, s2, result\_h, result\_m, result\_s);  
   
 /\* 座標変換のテスト \*/  
 printf("=== 座標変換 ===\n");  
 double r = 5.0, theta = M\_PI / 4; /\* 45度 \*/  
 double x, y;  
   
 polar\_to\_cartesian(r, theta, &x, &y);  
 printf("極座標(r=%.2f, θ=%.2f) → 直交座標(x=%.2f, y=%.2f)\n",   
 r, theta, x, y);  
   
 double x2 = 3.0, y2 = 4.0;  
 double r2, theta2;  
 cartesian\_to\_polar(x2, y2, &r2, &theta2);  
 printf("直交座標(x=%.2f, y=%.2f) → 極座標(r=%.2f, θ=%.2f)\n",   
 x2, y2, r2, theta2);  
   
 double mid\_x, mid\_y;  
 midpoint(0, 0, 10, 10, &mid\_x, &mid\_y);  
 printf("(0, 0)と(10, 10)の中点: (%.2f, %.2f)\n\n", mid\_x, mid\_y);  
   
 /\* 統計計算のテスト \*/  
 printf("=== 統計計算 ===\n");  
 double data[] = {85.5, 90.2, 78.3, 92.1, 88.7, 79.5, 91.3, 86.4, 83.2, 87.6};  
 int size = sizeof(data) / sizeof(data[0]);  
 double mean, variance, std\_dev;  
   
 basic\_statistics(data, size, &mean, &variance, &std\_dev);  
 printf("データ: ");  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 printf("%.1f ", data[i]);  
 }  
 printf("\n");  
 printf("平均: %.2f, 分散: %.2f, 標準偏差: %.2f\n", mean, variance, std\_dev);  
   
 double min, max, range;  
 min\_max\_range(data, size, &min, &max, &range);  
 printf("最小値: %.2f, 最大値: %.2f, 範囲: %.2f\n", min, max, range);  
   
 /\* データをソートしてパーセンタイル計算 \*/  
 double sorted\_data[10];  
 for (i = 0; i < size; i++)  
 {  
 sorted\_data[i] = data[i];  
 }  
 sort\_doubles(sorted\_data, size);  
   
 double q1, median, q3;  
 percentiles(sorted\_data, size, &q1, &median, &q3);  
 printf("第1四分位数: %.2f, 中央値: %.2f, 第3四分位数: %.2f\n", q1, median, q3);  
   
 return 0;  
}```  
  
## ex9\_5\_error\_handling.c  
  
```c  
/\*  
 \* 演習9-5: エラーハンドリング付き関数 - 解答例  
 \*   
 \* エラー処理を含む堅牢な関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <limits.h>  
#include <errno.h>  
  
/\* エラーコード定義 \*/  
#define SUCCESS 0  
#define ERROR\_NULL\_POINTER -1  
#define ERROR\_INVALID\_ARGUMENT -2  
#define ERROR\_OVERFLOW -3  
#define ERROR\_UNDERFLOW -4  
#define ERROR\_MEMORY\_ALLOCATION -5  
#define ERROR\_FILE\_NOT\_FOUND -6  
#define ERROR\_PERMISSION\_DENIED -7  
#define ERROR\_DIVIDE\_BY\_ZERO -8  
#define ERROR\_OUT\_OF\_RANGE -9  
  
/\* グローバルエラー状態（オプション） \*/  
static int g\_last\_error = SUCCESS;  
static char g\_error\_message[256] = "";  
  
/\* エラー処理のヘルパー関数 \*/  
void set\_error(int error\_code, const char \*message);  
int get\_last\_error(void);  
const char\* get\_error\_message(void);  
void clear\_error(void);  
const char\* error\_to\_string(int error\_code);  
  
/\* 数値計算関数（エラーチェック付き） \*/  
int safe\_add(int a, int b, int \*result);  
int safe\_multiply(int a, int b, int \*result);  
int safe\_divide(int dividend, int divisor, int \*result, int \*remainder);  
int safe\_power(int base, int exponent, long \*result);  
  
/\* 配列操作関数（エラーチェック付き） \*/  
int safe\_array\_access(int arr[], int size, int index, int \*value);  
int safe\_array\_sum(int arr[], int size, long \*sum);  
int safe\_array\_average(int arr[], int size, double \*average);  
  
/\* 文字列操作関数（エラーチェック付き） \*/  
int safe\_string\_copy(char \*dest, size\_t dest\_size, const char \*src);  
int safe\_string\_concat(char \*dest, size\_t dest\_size, const char \*src);  
int safe\_string\_to\_int(const char \*str, int \*value);  
  
/\* ファイル操作関数（エラーチェック付き） \*/  
int safe\_file\_read\_line(FILE \*file, char \*buffer, size\_t buffer\_size);  
int safe\_file\_write\_int(FILE \*file, int value);  
  
/\* エラーを設定する関数 \*/  
void set\_error(int error\_code, const char \*message)  
{  
 g\_last\_error = error\_code;  
 if (message != NULL)  
 {  
 strncpy(g\_error\_message, message, sizeof(g\_error\_message) - 1);  
 g\_error\_message[sizeof(g\_error\_message) - 1] = '\0';  
 }  
 else  
 {  
 g\_error\_message[0] = '\0';  
 }  
}  
  
/\* 最後のエラーコードを取得する関数 \*/  
int get\_last\_error(void)  
{  
 return g\_last\_error;  
}  
  
/\* エラーメッセージを取得する関数 \*/  
const char\* get\_error\_message(void)  
{  
 return g\_error\_message;  
}  
  
/\* エラー状態をクリアする関数 \*/  
void clear\_error(void)  
{  
 g\_last\_error = SUCCESS;  
 g\_error\_message[0] = '\0';  
}  
  
/\* エラーコードを文字列に変換する関数 \*/  
const char\* error\_to\_string(int error\_code)  
{  
 switch (error\_code)  
 {  
 case SUCCESS:  
 return "成功";  
 case ERROR\_NULL\_POINTER:  
 return "NULLポインターエラー";  
 case ERROR\_INVALID\_ARGUMENT:  
 return "無効な引数";  
 case ERROR\_OVERFLOW:  
 return "オーバーフロー";  
 case ERROR\_UNDERFLOW:  
 return "アンダーフロー";  
 case ERROR\_MEMORY\_ALLOCATION:  
 return "メモリ割り当てエラー";  
 case ERROR\_FILE\_NOT\_FOUND:  
 return "ファイルが見つかりません";  
 case ERROR\_PERMISSION\_DENIED:  
 return "アクセス権限がありません";  
 case ERROR\_DIVIDE\_BY\_ZERO:  
 return "ゼロ除算";  
 case ERROR\_OUT\_OF\_RANGE:  
 return "範囲外";  
 default:  
 return "不明なエラー";  
 }  
}  
  
/\* 安全な加算関数 \*/  
int safe\_add(int a, int b, int \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 /\* オーバーフローチェック \*/  
 if ((b > 0 && a > INT\_MAX - b) || (b < 0 && a < INT\_MIN - b))  
 {  
 set\_error(ERROR\_OVERFLOW, "加算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = a + b;  
 return SUCCESS;  
}  
  
/\* 安全な乗算関数 \*/  
int safe\_multiply(int a, int b, int \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 /\* オーバーフローチェック \*/  
 if (a > 0 && b > 0 && a > INT\_MAX / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a < 0 && b < 0 && a < INT\_MAX / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a > 0 && b < 0 && b < INT\_MIN / a)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a < 0 && b > 0 && a < INT\_MIN / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = a \* b;  
 return SUCCESS;  
}  
  
/\* 安全な除算関数 \*/  
int safe\_divide(int dividend, int divisor, int \*result, int \*remainder)  
{  
 if (result == NULL || remainder == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (divisor == 0)  
 {  
 set\_error(ERROR\_DIVIDE\_BY\_ZERO, "ゼロで除算");  
 return ERROR\_DIVIDE\_BY\_ZERO;  
 }  
   
 /\* INT\_MIN / -1 のオーバーフローチェック \*/  
 if (dividend == INT\_MIN && divisor == -1)  
 {  
 set\_error(ERROR\_OVERFLOW, "除算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = dividend / divisor;  
 \*remainder = dividend % divisor;  
 return SUCCESS;  
}  
  
/\* 安全な累乗関数 \*/  
int safe\_power(int base, int exponent, long \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (exponent < 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "負の指数はサポートされていません");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 long power = 1;  
 long temp\_base = base;  
   
 while (exponent > 0)  
 {  
 if (exponent & 1)  
 {  
 /\* オーバーフローチェック \*/  
 if (temp\_base > 0 && power > LONG\_MAX / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (temp\_base < 0 && power < LONG\_MIN / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 power \*= temp\_base;  
 }  
   
 exponent >>= 1;  
 if (exponent > 0)  
 {  
 /\* temp\_base \* temp\_base のオーバーフローチェック \*/  
 if (temp\_base > 0 && temp\_base > LONG\_MAX / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 temp\_base \*= temp\_base;  
 }  
 }  
   
 \*result = power;  
 return SUCCESS;  
}  
  
/\* 安全な配列アクセス関数 \*/  
int safe\_array\_access(int arr[], int size, int index, int \*value)  
{  
 if (arr == NULL || value == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または値ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 if (index < 0 || index >= size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "インデックスが範囲外");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 \*value = arr[index];  
 return SUCCESS;  
}  
  
/\* 安全な配列合計関数 \*/  
int safe\_array\_sum(int arr[], int size, long \*sum)  
{  
 if (arr == NULL || sum == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または合計ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 \*sum = 0;  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 /\* オーバーフローチェック \*/  
 if ((arr[i] > 0 && \*sum > LONG\_MAX - arr[i]) ||  
 (arr[i] < 0 && \*sum < LONG\_MIN - arr[i]))  
 {  
 set\_error(ERROR\_OVERFLOW, "配列合計でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 \*sum += arr[i];  
 }  
   
 return SUCCESS;  
}  
  
/\* 安全な配列平均関数 \*/  
int safe\_array\_average(int arr[], int size, double \*average)  
{  
 if (arr == NULL || average == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または平均ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 long sum;  
 int result = safe\_array\_sum(arr, size, &sum);  
 if (result != SUCCESS)  
 {  
 return result;  
 }  
   
 \*average = (double)sum / size;  
 return SUCCESS;  
}  
  
/\* 安全な文字列コピー関数 \*/  
int safe\_string\_copy(char \*dest, size\_t dest\_size, const char \*src)  
{  
 if (dest == NULL || src == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "送信先または送信元がNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (dest\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 size\_t src\_len = strlen(src);  
 if (src\_len >= dest\_size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "バッファサイズ不足");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 strcpy(dest, src);  
 return SUCCESS;  
}  
  
/\* 安全な文字列連結関数 \*/  
int safe\_string\_concat(char \*dest, size\_t dest\_size, const char \*src)  
{  
 if (dest == NULL || src == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "送信先または送信元がNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (dest\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 size\_t dest\_len = strlen(dest);  
 size\_t src\_len = strlen(src);  
   
 if (dest\_len + src\_len >= dest\_size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "バッファサイズ不足");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 strcat(dest, src);  
 return SUCCESS;  
}  
  
/\* 安全な文字列から整数への変換関数 \*/  
int safe\_string\_to\_int(const char \*str, int \*value)  
{  
 if (str == NULL || value == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "文字列または値ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 char \*endptr;  
 errno = 0;  
 long result = strtol(str, &endptr, 10);  
   
 /\* 変換エラーチェック \*/  
 if (errno == ERANGE || result > INT\_MAX || result < INT\_MIN)  
 {  
 set\_error(ERROR\_OVERFLOW, "整数変換でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 if (endptr == str || \*endptr != '\0')  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "無効な数値形式");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 \*value = (int)result;  
 return SUCCESS;  
}  
  
/\* 安全なファイル読み込み関数 \*/  
int safe\_file\_read\_line(FILE \*file, char \*buffer, size\_t buffer\_size)  
{  
 if (file == NULL || buffer == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "ファイルまたはバッファがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (buffer\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 if (fgets(buffer, buffer\_size, file) == NULL)  
 {  
 if (feof(file))  
 {  
 set\_error(ERROR\_UNDERFLOW, "ファイルの終端に達しました");  
 return ERROR\_UNDERFLOW;  
 }  
 else  
 {  
 set\_error(ERROR\_FILE\_NOT\_FOUND, "ファイル読み込みエラー");  
 return ERROR\_FILE\_NOT\_FOUND;  
 }  
 }  
   
 return SUCCESS;  
}  
  
/\* 安全なファイル書き込み関数 \*/  
int safe\_file\_write\_int(FILE \*file, int value)  
{  
 if (file == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "ファイルポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (fprintf(file, "%d\n", value) < 0)  
 {  
 set\_error(ERROR\_PERMISSION\_DENIED, "ファイル書き込みエラー");  
 return ERROR\_PERMISSION\_DENIED;  
 }  
   
 return SUCCESS;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 int result;  
 int int\_result;  
 long long\_result;  
 double double\_result;  
 char buffer[100];  
   
 printf("=== エラーハンドリング付き関数のテスト ===\n\n");  
   
 /\* 数値計算のテスト \*/  
 printf("=== 数値計算 ===\n");  
   
 /\* 正常な加算 \*/  
 result = safe\_add(100, 200, &int\_result);  
 printf("100 + 200 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* オーバーフローする加算 \*/  
 result = safe\_add(INT\_MAX, 1, &int\_result);  
 printf("INT\_MAX + 1 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 除算のテスト \*/  
 int remainder;  
 result = safe\_divide(17, 5, &int\_result, &remainder);  
 printf("17 ÷ 5 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d 余り %d (成功)\n", int\_result, remainder);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* ゼロ除算 \*/  
 result = safe\_divide(10, 0, &int\_result, &remainder);  
 printf("10 ÷ 0 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 累乗のテスト \*/  
 result = safe\_power(2, 10, &long\_result);  
 printf("2^10 = ");  
 if (result == SUCCESS)  
 {  
 printf("%ld (成功)\n", long\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
 printf("\n");  
   
 /\* 配列操作のテスト \*/  
 printf("=== 配列操作 ===\n");  
 int test\_array[] = {10, 20, 30, 40, 50};  
 int array\_size = 5;  
   
 /\* 正常な配列アクセス \*/  
 result = safe\_array\_access(test\_array, array\_size, 2, &int\_result);  
 printf("配列[2] = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* 範囲外アクセス \*/  
 result = safe\_array\_access(test\_array, array\_size, 10, &int\_result);  
 printf("配列[10] = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 配列平均 \*/  
 result = safe\_array\_average(test\_array, array\_size, &double\_result);  
 printf("配列の平均 = ");  
 if (result == SUCCESS)  
 {  
 printf("%.2f (成功)\n", double\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
 printf("\n");  
   
 /\* 文字列操作のテスト \*/  
 printf("=== 文字列操作 ===\n");  
   
 /\* 正常な文字列コピー \*/  
 result = safe\_string\_copy(buffer, sizeof(buffer), "Hello, World!");  
 printf("文字列コピー: ");  
 if (result == SUCCESS)  
 {  
 printf("\"%s\" (成功)\n", buffer);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* バッファオーバーフロー防止 \*/  
 char small\_buffer[5];  
 result = safe\_string\_copy(small\_buffer, sizeof(small\_buffer), "This is too long");  
 printf("小さいバッファへのコピー: ");  
 if (result == SUCCESS)  
 {  
 printf("\"%s\" (成功)\n", small\_buffer);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 文字列から整数への変換 \*/  
 result = safe\_string\_to\_int("12345", &int\_result);  
 printf("\"12345\" を整数に変換: ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* 無効な文字列 \*/  
 result = safe\_string\_to\_int("abc123", &int\_result);  
 printf("\"abc123\" を整数に変換: ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 printf("\n");  
   
 /\* エラー処理のサマリー \*/  
 printf("=== エラー処理のサマリー ===\n");  
 printf("エラーハンドリングにより、以下の問題を防ぐことができました：\n");  
 printf("- オーバーフロー/アンダーフロー\n");  
 printf("- ゼロ除算\n");  
 printf("- 範囲外アクセス\n");  
 printf("- バッファオーバーフロー\n");  
 printf("- 無効な入力\n");  
 printf("- NULLポインター参照\n");  
   
 return 0;  
}```  
  
## ex9\_6\_struct\_functions.c  
  
```c  
/\*  
 \* 演習9-6: 構造体を使った関数 - 解答例  
 \*   
 \* 構造体を引数や戻り値として使用する関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
  
/\* 構造体定義 \*/  
/\* 商品情報 \*/  
typedef struct {  
 int id;  
 char name[50];  
 double price;  
 int stock;  
} Product;  
  
/\* 学生情報 \*/  
typedef struct {  
 int id;  
 char name[50];  
 double score;  
 char grade;  
} Student;  
  
/\* 日付 \*/  
typedef struct {  
 int year;  
 int month;  
 int day;  
} Date;  
  
/\* 座標点 \*/  
typedef struct {  
 double x;  
 double y;  
} Point;  
  
/\* 矩形 \*/  
typedef struct {  
 Point top\_left;  
 Point bottom\_right;  
} Rectangle;  
  
/\* 関数プロトタイプ \*/  
/\* 商品管理関数 \*/  
Product create\_product(int id, const char \*name, double price, int stock);  
void print\_product(Product p);  
double calculate\_total\_value(Product p);  
int compare\_products\_by\_price(Product p1, Product p2);  
void update\_stock(Product \*p, int quantity);  
void apply\_discount(Product \*p, double discount\_percent);  
  
/\* 学生管理関数 \*/  
Student create\_student(int id, const char \*name, double score);  
void print\_student(Student s);  
void update\_score(Student \*s, double new\_score);  
char calculate\_grade(double score);  
Student find\_best\_student(Student students[], int count);  
void sort\_students\_by\_score(Student students[], int count);  
  
/\* 日付操作関数 \*/  
Date create\_date(int year, int month, int day);  
void print\_date(Date d);  
int is\_valid\_date(Date d);  
int is\_leap\_year(int year);  
int days\_in\_month(int year, int month);  
int date\_difference(Date d1, Date d2);  
int compare\_dates(Date d1, Date d2);  
  
/\* 座標・図形関数 \*/  
Point create\_point(double x, double y);  
double point\_distance(Point p1, Point p2);  
Rectangle create\_rectangle(Point top\_left, Point bottom\_right);  
double rectangle\_area(Rectangle r);  
double rectangle\_perimeter(Rectangle r);  
int point\_in\_rectangle(Point p, Rectangle r);  
  
/\* 商品を作成する関数 \*/  
Product create\_product(int id, const char \*name, double price, int stock)  
{  
 Product p;  
 p.id = id;  
 strncpy(p.name, name, sizeof(p.name) - 1);  
 p.name[sizeof(p.name) - 1] = '\0';  
 p.price = price;  
 p.stock = stock;  
 return p;  
}  
  
/\* 商品情報を表示する関数 \*/  
void print\_product(Product p)  
{  
 printf("商品ID: %d\n", p.id);  
 printf("商品名: %s\n", p.name);  
 printf("価格: %.2f円\n", p.price);  
 printf("在庫: %d個\n", p.stock);  
}  
  
/\* 商品の総価値を計算する関数 \*/  
double calculate\_total\_value(Product p)  
{  
 return p.price \* p.stock;  
}  
  
/\* 商品を価格で比較する関数 \*/  
int compare\_products\_by\_price(Product p1, Product p2)  
{  
 if (p1.price < p2.price) return -1;  
 if (p1.price > p2.price) return 1;  
 return 0;  
}  
  
/\* 在庫を更新する関数 \*/  
void update\_stock(Product \*p, int quantity)  
{  
 if (p != NULL)  
 {  
 p->stock += quantity;  
 if (p->stock < 0)  
 {  
 p->stock = 0;  
 }  
 }  
}  
  
/\* 割引を適用する関数 \*/  
void apply\_discount(Product \*p, double discount\_percent)  
{  
 if (p != NULL && discount\_percent >= 0.0 && discount\_percent <= 100.0)  
 {  
 p->price \*= (1.0 - discount\_percent / 100.0);  
 }  
}  
  
/\* 学生を作成する関数 \*/  
Student create\_student(int id, const char \*name, double score)  
{  
 Student s;  
 s.id = id;  
 strncpy(s.name, name, sizeof(s.name) - 1);  
 s.name[sizeof(s.name) - 1] = '\0';  
 s.score = score;  
 s.grade = calculate\_grade(score);  
 return s;  
}  
  
/\* 学生情報を表示する関数 \*/  
void print\_student(Student s)  
{  
 printf("学生ID: %d\n", s.id);  
 printf("氏名: %s\n", s.name);  
 printf("得点: %.1f点\n", s.score);  
 printf("成績: %c\n", s.grade);  
}  
  
/\* 学生の成績を更新する関数 \*/  
void update\_score(Student \*s, double new\_score)  
{  
 if (s != NULL && new\_score >= 0.0 && new\_score <= 100.0)  
 {  
 s->score = new\_score;  
 s->grade = calculate\_grade(new\_score);  
 }  
}  
  
/\* 得点から成績を計算する関数 \*/  
char calculate\_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';  
}  
  
/\* 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)  
{  
 Student best = students[0];  
 int i;  
   
 for (i = 1; i < count; i++)  
 {  
 if (students[i].score > best.score)  
 {  
 best = students[i];  
 }  
 }  
   
 return best;  
}  
  
/\* 学生を得点順にソートする関数 \*/  
void sort\_students\_by\_score(Student students[], int count)  
{  
 int i, j;  
 Student temp;  
   
 for (i = 0; i < count - 1; i++)  
 {  
 for (j = 0; j < count - 1 - i; j++)  
 {  
 if (students[j].score < students[j + 1].score)  
 {  
 temp = students[j];  
 students[j] = students[j + 1];  
 students[j + 1] = temp;  
 }  
 }  
 }  
}  
  
/\* 日付を作成する関数 \*/  
Date create\_date(int year, int month, int day)  
{  
 Date d;  
 d.year = year;  
 d.month = month;  
 d.day = day;  
 return d;  
}  
  
/\* 日付を表示する関数 \*/  
void print\_date(Date d)  
{  
 printf("%04d年%02d月%02d日", d.year, d.month, d.day);  
}  
  
/\* 日付の妥当性をチェックする関数 \*/  
int is\_valid\_date(Date d)  
{  
 if (d.year < 1 || d.year > 9999)  
 {  
 return 0;  
 }  
   
 if (d.month < 1 || d.month > 12)  
 {  
 return 0;  
 }  
   
 if (d.day < 1 || d.day > days\_in\_month(d.year, d.month))  
 {  
 return 0;  
 }  
   
 return 1;  
}  
  
/\* うるう年かどうかを判定する関数 \*/  
int is\_leap\_year(int year)  
{  
 return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);  
}  
  
/\* 月の日数を返す関数 \*/  
int days\_in\_month(int year, int month)  
{  
 int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
   
 if (month < 1 || month > 12)  
 {  
 return 0;  
 }  
   
 if (month == 2 && is\_leap\_year(year))  
 {  
 return 29;  
 }  
   
 return days[month - 1];  
}  
  
/\* 2つの日付の差を計算する関数（簡易版） \*/  
int date\_difference(Date d1, Date d2)  
{  
 /\* より正確な実装には、各月の日数を考慮した計算が必要 \*/  
 /\* ここでは簡易的に年と月を日数に変換 \*/  
 int days1 = d1.year \* 365 + d1.month \* 30 + d1.day;  
 int days2 = d2.year \* 365 + d2.month \* 30 + d2.day;  
   
 return days1 - days2;  
}  
  
/\* 日付を比較する関数 \*/  
int compare\_dates(Date d1, Date d2)  
{  
 if (d1.year != d2.year)  
 {  
 return d1.year - d2.year;  
 }  
   
 if (d1.month != d2.month)  
 {  
 return d1.month - d2.month;  
 }  
   
 return d1.day - d2.day;  
}  
  
/\* 点を作成する関数 \*/  
Point create\_point(double x, double y)  
{  
 Point p;  
 p.x = x;  
 p.y = y;  
 return p;  
}  
  
/\* 2点間の距離を計算する関数 \*/  
double point\_distance(Point p1, Point p2)  
{  
 double dx = p2.x - p1.x;  
 double dy = p2.y - p1.y;  
 return sqrt(dx \* dx + dy \* dy);  
}  
  
/\* 矩形を作成する関数 \*/  
Rectangle create\_rectangle(Point top\_left, Point bottom\_right)  
{  
 Rectangle r;  
 r.top\_left = top\_left;  
 r.bottom\_right = bottom\_right;  
 return r;  
}  
  
/\* 矩形の面積を計算する関数 \*/  
double rectangle\_area(Rectangle r)  
{  
 double width = r.bottom\_right.x - r.top\_left.x;  
 double height = r.top\_left.y - r.bottom\_right.y;  
 return fabs(width \* height);  
}  
  
/\* 矩形の周囲長を計算する関数 \*/  
double rectangle\_perimeter(Rectangle r)  
{  
 double width = fabs(r.bottom\_right.x - r.top\_left.x);  
 double height = fabs(r.top\_left.y - r.bottom\_right.y);  
 return 2.0 \* (width + height);  
}  
  
/\* 点が矩形内にあるかを判定する関数 \*/  
int point\_in\_rectangle(Point p, Rectangle r)  
{  
 return p.x >= r.top\_left.x && p.x <= r.bottom\_right.x &&  
 p.y <= r.top\_left.y && p.y >= r.bottom\_right.y;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 printf("=== 構造体を使った関数のテスト ===\n\n");  
   
 /\* 商品管理のテスト \*/  
 printf("=== 商品管理 ===\n");  
 Product p1 = create\_product(1001, "ノートパソコン", 98000.0, 5);  
 Product p2 = create\_product(1002, "マウス", 2980.0, 20);  
   
 print\_product(p1);  
 printf("総価値: %.2f円\n\n", calculate\_total\_value(p1));  
   
 print\_product(p2);  
 printf("総価値: %.2f円\n\n", calculate\_total\_value(p2));  
   
 printf("価格比較: ");  
 int cmp = compare\_products\_by\_price(p1, p2);  
 if (cmp > 0)  
 {  
 printf("商品1の方が高い\n");  
 }  
 else if (cmp < 0)  
 {  
 printf("商品2の方が高い\n");  
 }  
 else  
 {  
 printf("同じ価格\n");  
 }  
   
 printf("\n商品1に20%%割引を適用\n");  
 apply\_discount(&p1, 20.0);  
 print\_product(p1);  
 printf("\n");  
   
 /\* 学生管理のテスト \*/  
 printf("=== 学生管理 ===\n");  
 Student students[] = {  
 create\_student(2001, "田中太郎", 85.5),  
 create\_student(2002, "鈴木花子", 92.0),  
 create\_student(2003, "佐藤次郎", 78.3),  
 create\_student(2004, "山田美咲", 88.5)  
 };  
 int student\_count = 4;  
   
 int i;  
 printf("学生一覧:\n");  
 for (i = 0; i < student\_count; i++)  
 {  
 printf("--- 学生%d ---\n", i + 1);  
 print\_student(students[i]);  
 }  
   
 printf("\n最高得点者:\n");  
 Student best = find\_best\_student(students, student\_count);  
 print\_student(best);  
   
 printf("\n得点順にソート後:\n");  
 sort\_students\_by\_score(students, student\_count);  
 for (i = 0; i < student\_count; i++)  
 {  
 printf("%d位: %s (%.1f点)\n", i + 1, students[i].name, students[i].score);  
 }  
 printf("\n");  
   
 /\* 日付操作のテスト \*/  
 printf("=== 日付操作 ===\n");  
 Date d1 = create\_date(2024, 3, 15);  
 Date d2 = create\_date(2024, 12, 25);  
 Date d3 = create\_date(2024, 2, 29); /\* うるう年 \*/  
 Date d4 = create\_date(2023, 2, 29); /\* 無効な日付 \*/  
   
 printf("日付1: ");  
 print\_date(d1);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d1) ? "OK" : "NG");  
   
 printf("日付2: ");  
 print\_date(d2);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d2) ? "OK" : "NG");  
   
 printf("日付3: ");  
 print\_date(d3);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d3) ? "OK" : "NG");  
   
 printf("日付4: ");  
 print\_date(d4);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d4) ? "OK" : "NG");  
   
 printf("\n2024年はうるう年? %s\n", is\_leap\_year(2024) ? "はい" : "いいえ");  
 printf("2023年はうるう年? %s\n", is\_leap\_year(2023) ? "はい" : "いいえ");  
   
 printf("\n日付の比較:\n");  
 int date\_cmp = compare\_dates(d1, d2);  
 if (date\_cmp < 0)  
 {  
 printf("日付1の方が早い\n");  
 }  
 else if (date\_cmp > 0)  
 {  
 printf("日付2の方が早い\n");  
 }  
 else  
 {  
 printf("同じ日付\n");  
 }  
 printf("\n");  
   
 /\* 座標・図形のテスト \*/  
 printf("=== 座標・図形 ===\n");  
 Point p\_1 = create\_point(0.0, 0.0);  
 Point p\_2 = create\_point(3.0, 4.0);  
 Point p\_3 = create\_point(5.0, 5.0);  
   
 printf("点1: (%.1f, %.1f)\n", p\_1.x, p\_1.y);  
 printf("点2: (%.1f, %.1f)\n", p\_2.x, p\_2.y);  
 printf("点1と点2の距離: %.2f\n", point\_distance(p\_1, p\_2));  
   
 Rectangle rect = create\_rectangle(create\_point(0.0, 10.0), create\_point(10.0, 0.0));  
 printf("\n矩形: 左上(%.1f, %.1f) - 右下(%.1f, %.1f)\n",  
 rect.top\_left.x, rect.top\_left.y,  
 rect.bottom\_right.x, rect.bottom\_right.y);  
 printf("面積: %.2f\n", rectangle\_area(rect));  
 printf("周囲長: %.2f\n", rectangle\_perimeter(rect));  
   
 printf("\n点の位置判定:\n");  
 printf("点2(%.1f, %.1f)は矩形内? %s\n",   
 p\_2.x, p\_2.y, point\_in\_rectangle(p\_2, rect) ? "はい" : "いいえ");  
 printf("点3(%.1f, %.1f)は矩形内? %s\n",   
 p\_3.x, p\_3.y, point\_in\_rectangle(p\_3, rect) ? "はい" : "いいえ");  
   
 return 0;  
}```  
  
## ex9\_7\_recursive\_functions.c  
  
```c  
/\*  
 \* ファイル名: ex9\_7\_recursive\_functions.c  
 \* 演習9-7: 再帰関数  
 \* 説明: 階乗、フィボナッチ、ユークリッドの互除法などの再帰関数の実装  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
long factorial\_recursive(int n);  
int fibonacci\_recursive(int n);  
int gcd\_recursive(int a, int b);  
int is\_palindrome\_recursive(const char str[], int start, int end);  
void reverse\_string\_recursive(char str[], int start, int end);  
int array\_sum\_recursive(int arr[], int size);  
int find\_max\_recursive(int arr[], int size);  
  
/\* 階乗を計算する再帰関数 \*/  
long factorial\_recursive(int n)  
{  
 /\* ベースケース \*/  
 if (n <= 1)  
 {  
 return 1;  
 }  
   
 /\* 再帰ケース \*/  
 return n \* factorial\_recursive(n - 1);  
}  
  
/\* フィボナッチ数列を計算する再帰関数 \*/  
int fibonacci\_recursive(int n)  
{  
 /\* ベースケース \*/  
 if (n <= 0)  
 {  
 return 0;  
 }  
 if (n == 1)  
 {  
 return 1;  
 }  
   
 /\* 再帰ケース \*/  
 return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2);  
}  
  
/\* 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)  
{  
 /\* ベースケース \*/  
 if (b == 0)  
 {  
 return a;  
 }  
   
 /\* 再帰ケース \*/  
 return gcd\_recursive(b, a % b);  
}  
  
/\* 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)  
{  
 /\* ベースケース: 文字が交差したか同じ位置になった場合 \*/  
 if (start >= end)  
 {  
 return 1; /\* 回文 \*/  
 }  
   
 /\* 現在の位置の文字が異なる場合 \*/  
 if (str[start] != str[end])  
 {  
 return 0; /\* 回文ではない \*/  
 }  
   
 /\* 再帰ケース: 内側の文字列をチェック \*/  
 return is\_palindrome\_recursive(str, start + 1, end - 1);  
}  
  
/\* 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)  
{  
 char temp;  
   
 /\* ベースケース \*/  
 if (start >= end)  
 {  
 return;  
 }  
   
 /\* 文字を交換 \*/  
 temp = str[start];  
 str[start] = str[end];  
 str[end] = temp;  
   
 /\* 再帰ケース \*/  
 reverse\_string\_recursive(str, start + 1, end - 1);  
}  
  
/\* 配列の合計を再帰的に計算する関数 \*/  
int array\_sum\_recursive(int arr[], int size)  
{  
 /\* ベースケース \*/  
 if (size <= 0)  
 {  
 return 0;  
 }  
   
 /\* 再帰ケース \*/  
 return arr[size - 1] + array\_sum\_recursive(arr, size - 1);  
}  
  
/\* 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)  
{  
 int max\_of\_rest;  
   
 /\* ベースケース \*/  
 if (size == 1)  
 {  
 return arr[0];  
 }  
   
 /\* 再帰ケース \*/  
 max\_of\_rest = find\_max\_recursive(arr, size - 1);  
   
 /\* 現在の要素と残りの最大値を比較 \*/  
 return (arr[size - 1] > max\_of\_rest) ? arr[size - 1] : max\_of\_rest;  
}  
  
/\* 文字列の長さを求める補助関数 \*/  
int my\_strlen(const char str[])  
{  
 int len = 0;  
 while (str[len] != '\0')  
 {  
 len++;  
 }  
 return len;  
}  
  
int main(void)  
{  
 /\* C90: すべての変数を関数先頭で宣言 \*/  
 int factorial\_tests[] = {0, 1, 5, 10, 12};  
 int num\_factorial\_tests = sizeof(factorial\_tests) / sizeof(factorial\_tests[0]);  
 int gcd\_pairs[][2] = {{48, 18}, {60, 48}, {17, 13}, {100, 75}};  
 int num\_gcd\_pairs = sizeof(gcd\_pairs) / sizeof(gcd\_pairs[0]);  
 char\* palindrome\_tests[] = {"racecar", "hello", "madam", "abcba", "test"};  
 int num\_palindrome\_tests = sizeof(palindrome\_tests) / sizeof(palindrome\_tests[0]);  
 char test\_strings[][20] = {"hello", "world", "recursive", "function"};  
 int num\_strings = sizeof(test\_strings) / sizeof(test\_strings[0]);  
 int test\_array1[] = {1, 2, 3, 4, 5};  
 int test\_array2[] = {10, 20, 30, 40};  
 int test\_array3[] = {7, 14, 21, 28, 35, 42};  
 int i;  
   
 printf("=== 再帰関数のデモ ===\n\n");  
  
 /\* デモ1: 階乗計算 \*/  
 printf("1. 階乗計算\n");  
   
 for (i = 0; i < num\_factorial\_tests; i++)  
 {  
 int n = factorial\_tests[i];  
 printf(" %d! = %ld\n", n, factorial\_recursive(n));  
 }  
 printf("\n");  
  
 /\* デモ2: フィボナッチ数列 \*/  
 printf("2. フィボナッチ数列\n");  
 printf(" 最初の15項: ");  
 for (i = 0; i < 15; i++)  
 {  
 printf("%d ", fibonacci\_recursive(i));  
 }  
 printf("\n\n");  
  
 /\* デモ3: 最大公約数 \*/  
 printf("3. 最大公約数（ユークリッドの互除法）\n");  
   
 for (i = 0; i < num\_gcd\_pairs; i++)  
 {  
 int a = gcd\_pairs[i][0];  
 int b = gcd\_pairs[i][1];  
 printf(" gcd(%d, %d) = %d\n", a, b, gcd\_recursive(a, b));  
 }  
 printf("\n");  
  
 /\* デモ4: 回文判定 \*/  
 printf("4. 回文判定\n");  
   
 for (i = 0; i < num\_palindrome\_tests; i++)  
 {  
 char\* str = palindrome\_tests[i];  
 int len = my\_strlen(str);  
 int is\_pal = is\_palindrome\_recursive(str, 0, len - 1);  
 printf(" \"%s\" -> %s\n", str, is\_pal ? "回文です" : "回文ではありません");  
 }  
 printf("\n");  
  
 /\* デモ5: 文字列の逆順 \*/  
 printf("5. 文字列の逆順\n");  
   
 for (i = 0; i < num\_strings; i++)  
 {  
 char original[20];  
 int j;  
 int len;  
   
 /\* 元の文字列をコピー \*/  
 for (j = 0; test\_strings[i][j] != '\0'; j++)  
 {  
 original[j] = test\_strings[i][j];  
 }  
 original[j] = '\0';  
   
 len = my\_strlen(test\_strings[i]);  
 reverse\_string\_recursive(test\_strings[i], 0, len - 1);  
 printf(" \"%s\" -> \"%s\"\n", original, test\_strings[i]);  
 }  
 printf("\n");  
  
 /\* デモ6: 配列の合計 \*/  
 printf("6. 配列の合計（再帰）\n");  
   
 printf(" 配列 {1, 2, 3, 4, 5} の合計: %d\n",   
 array\_sum\_recursive(test\_array1, 5));  
 printf(" 配列 {10, 20, 30, 40} の合計: %d\n",   
 array\_sum\_recursive(test\_array2, 4));  
 printf(" 配列 {7, 14, 21, 28, 35, 42} の合計: %d\n",   
 array\_sum\_recursive(test\_array3, 6));  
 printf("\n");  
  
 /\* デモ7: 配列の最大値 \*/  
 printf("7. 配列の最大値（再帰）\n");  
 printf(" 配列 {1, 2, 3, 4, 5} の最大値: %d\n",   
 find\_max\_recursive(test\_array1, 5));  
 printf(" 配列 {10, 20, 30, 40} の最大値: %d\n",   
 find\_max\_recursive(test\_array2, 4));  
 printf(" 配列 {7, 14, 21, 28, 35, 42} の最大値: %d\n",   
 find\_max\_recursive(test\_array3, 6));  
 printf("\n");  
  
 printf("=== 再帰関数デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 再帰関数の基本構造（ベースケースと再帰ケース）  
2. 数学的計算での再帰の応用  
3. 文字列処理での再帰の活用  
4. 配列処理での再帰的アプローチ  
5. 再帰の効率性と限界の理解  
  
実装のポイント:  
- C90準拠のため、変数は関数先頭で宣言  
- ベースケースの重要性  
- スタックオーバーフローの注意  
- 再帰よりも反復が効率的な場合の判断  
\*/

## ex9\_8\_advanced\_strings.c

```c / ファイル名: ex9\_8\_advanced\_strings.c \* 演習9-8: 高度な文字列処理 \* 説明: 文字列の解析、パターンマッチング、フォーマット処理などの高度な関数群 \* 規格: C90準拠 \*/ #include <stdio.h>

/\* 関数プロトタイプ \*/ int tokenize(char str[], char tokens[][50], char delimiter); int check\_brackets(const char str[]); int extract\_numbers(const char str[], int numbers[], int max\_count); int wildcard\_match(const char str[], const char pattern[]); int string\_replace(char str[], const char old\_substr[], const char new\_substr[]); void center\_string(char result[], const char str[], int width); int parse\_csv\_line(const char line[], char fields[][100], int max\_fields);

/\* 補助関数 \*/ int my\_strlen(const char str[]); void my\_strcpy(char dest[], const char src[]); int my\_strncmp(const char str1[], const char str2[], int n); int is\_digit(char c);

/\* 文字列をトークンに分割する関数 \*/ int tokenize(char str[], char tokens[][50], char delimiter) { int token\_count = 0; int i = 0; int token\_start = 0; int token\_len; int j;

while (str[i] != '\0' && token\_count < 20) /\* 最大20トークン \*/  
{  
 if (str[i] == delimiter)  
 {  
 /\* トークンの終了 \*/  
 token\_len = i - token\_start;  
 if (token\_len > 0)  
 {  
 /\* トークンをコピー \*/  
 for (j = 0; j < token\_len && j < 49; j++)  
 {  
 tokens[token\_count][j] = str[token\_start + j];  
 }  
 tokens[token\_count][j] = '\0';  
 token\_count++;  
 }  
 token\_start = i + 1;  
 }  
 i++;  
}  
  
/\* 最後のトークン \*/  
token\_len = i - token\_start;  
if (token\_len > 0 && token\_count < 20)  
{  
 for (j = 0; j < token\_len && j < 49; j++)  
 {  
 tokens[token\_count][j] = str[token\_start + j];  
 }  
 tokens[token\_count][j] = '\0';  
 token\_count++;  
}  
  
return token\_count;

}

/\* 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[]) { int round\_count = 0; /* ( ) */ int square\_count = 0; /* [ ] */ int curly\_count = 0; /* { } \*/ int i = 0;

while (str[i] != '\0')  
{  
 switch (str[i])  
 {  
 case '(':  
 round\_count++;  
 break;  
 case ')':  
 round\_count--;  
 if (round\_count < 0) return 0; /\* 閉じ括弧が多い \*/  
 break;  
 case '[':  
 square\_count++;  
 break;  
 case ']':  
 square\_count--;  
 if (square\_count < 0) return 0;  
 break;  
 case '{':  
 curly\_count++;  
 break;  
 case '}':  
 curly\_count--;  
 if (curly\_count < 0) return 0;  
 break;  
 }  
 i++;  
}  
  
/\* すべての括弧がバランスしているかチェック \*/  
return (round\_count == 0 && square\_count == 0 && curly\_count == 0);

}

/\* 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) { int count = 0; int i = 0; int current\_number = 0; int in\_number = 0; int is\_negative = 0;

while (str[i] != '\0' && count < max\_count)  
{  
 if (is\_digit(str[i]))  
 {  
 if (!in\_number)  
 {  
 in\_number = 1;  
 current\_number = 0;  
 is\_negative = 0;  
   
 /\* 負の数のチェック \*/  
 if (i > 0 && str[i - 1] == '-')  
 {  
 is\_negative = 1;  
 }  
 }  
 current\_number = current\_number \* 10 + (str[i] - '0');  
 }  
 else  
 {  
 if (in\_number)  
 {  
 /\* 数値の終了 \*/  
 numbers[count] = is\_negative ? -current\_number : current\_number;  
 count++;  
 in\_number = 0;  
 }  
 }  
 i++;  
}  
  
/\* 文字列の最後が数値の場合 \*/  
if (in\_number && count < max\_count)  
{  
 numbers[count] = is\_negative ? -current\_number : current\_number;  
 count++;  
}  
  
return count;

}

/\* 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[]) { int str\_i = 0; int pat\_i = 0; int str\_len = my\_strlen(str); int pat\_len = my\_strlen(pattern);

while (str\_i < str\_len && pat\_i < pat\_len)  
{  
 if (pattern[pat\_i] == '\*')  
 {  
 /\* \* は0文字以上にマッチ \*/  
 pat\_i++;  
 if (pat\_i == pat\_len) return 1; /\* パターンの最後が \* \*/  
   
 /\* 次の文字が見つかるまでスキップ \*/  
 while (str\_i < str\_len)  
 {  
 if (wildcard\_match(&str[str\_i], &pattern[pat\_i]))  
 {  
 return 1;  
 }  
 str\_i++;  
 }  
 return 0;  
 }  
 else if (pattern[pat\_i] == '?' || pattern[pat\_i] == str[str\_i])  
 {  
 /\* ? は任意の1文字、または文字が一致 \*/  
 str\_i++;  
 pat\_i++;  
 }  
 else  
 {  
 return 0; /\* 不一致 \*/  
 }  
}  
  
/\* 残りのパターンがすべて \* かチェック \*/  
while (pat\_i < pat\_len && pattern[pat\_i] == '\*')  
{  
 pat\_i++;  
}  
  
return (str\_i == str\_len && pat\_i == pat\_len);

}

/\* 文字列の置換関数（簡単版） */ int string\_replace(char str[], const char old\_substr[], const char new\_substr[]) { char temp[1000]; /* 作業用バッファ \*/ int old\_len = my\_strlen(old\_substr); int new\_len = my\_strlen(new\_substr); int str\_len = my\_strlen(str); int i = 0; int temp\_i = 0; int replacements = 0;

while (i < str\_len)  
{  
 if (my\_strncmp(&str[i], old\_substr, old\_len) == 0)  
 {  
 /\* 置換対象が見つかった \*/  
 int j;  
 for (j = 0; j < new\_len; j++)  
 {  
 temp[temp\_i++] = new\_substr[j];  
 }  
 i += old\_len;  
 replacements++;  
 }  
 else  
 {  
 temp[temp\_i++] = str[i];  
 i++;  
 }  
}  
  
temp[temp\_i] = '\0';  
my\_strcpy(str, temp);  
  
return replacements;

}

/\* 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) { int str\_len = my\_strlen(str); int padding = (width - str\_len) / 2; int i;

if (str\_len >= width)  
{  
 /\* 文字列が幅より長い場合はそのままコピー \*/  
 my\_strcpy(result, str);  
 return;  
}  
  
/\* 左側のパディング \*/  
for (i = 0; i < padding; i++)  
{  
 result[i] = ' ';  
}  
  
/\* 文字列をコピー \*/  
for (i = 0; i < str\_len; i++)  
{  
 result[padding + i] = str[i];  
}  
  
/\* 右側のパディング \*/  
for (i = padding + str\_len; i < width; i++)  
{  
 result[i] = ' ';  
}  
  
result[width] = '\0';

}

/\* 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) { int field\_count = 0; int i = 0; int field\_start = 0; int in\_quotes = 0; int field\_len; int j;

while (line[i] != '\0' && field\_count < max\_fields)  
{  
 if (line[i] == '"')  
 {  
 in\_quotes = !in\_quotes;  
 }  
 else if (line[i] == ',' && !in\_quotes)  
 {  
 /\* フィールドの終了 \*/  
 field\_len = i - field\_start;  
 for (j = 0; j < field\_len && j < 99; j++)  
 {  
 fields[field\_count][j] = line[field\_start + j];  
 }  
 fields[field\_count][j] = '\0';  
 field\_count++;  
 field\_start = i + 1;  
 }  
 i++;  
}  
  
/\* 最後のフィールド \*/  
if (field\_count < max\_fields)  
{  
 field\_len = i - field\_start;  
 for (j = 0; j < field\_len && j < 99; j++)  
 {  
 fields[field\_count][j] = line[field\_start + j];  
 }  
 fields[field\_count][j] = '\0';  
 field\_count++;  
}  
  
return field\_count;

}

/\* 補助関数の実装 \*/ int my\_strlen(const char str[]) { int len = 0; while (str[len] != ‘\0’) len++; return len; }

void my\_strcpy(char dest[], const char src[]) { int i = 0; while (src[i] != ‘\0’) { dest[i] = src[i]; i++; } dest[i] = ‘\0’; }

int my\_strncmp(const char str1[], const char str2[], int n) { int i; for (i = 0; i < n; i++) { if (str1[i] != str2[i]) { return str1[i] - str2[i]; } if (str1[i] == ‘\0’) { return 0; } } return 0; }

int is\_digit(char c) { return (c >= ‘0’ && c <= ‘9’); }

int main(void) { printf(“=== 高度な文字列処理のデモ ===”);

/\* デモ1: 文字列の分割 \*/  
printf("1. 文字列の分割（トークン化）\n");  
char test\_string1[] = "apple,banana,orange,grape";  
char tokens[20][50];  
int token\_count = tokenize(test\_string1, tokens, ',');  
int i, j;  
  
printf(" 文字列: \"%s\"\n", "apple,banana,orange,grape");  
printf(" 分割結果: ");  
for (i = 0; i < token\_count; i++)  
{  
 printf("\"%s\"", tokens[i]);  
 if (i < token\_count - 1) printf(", ");  
}  
printf("\n\n");  
  
/\* デモ2: 括弧のバランスチェック \*/  
printf("2. 括弧のバランスチェック\n");  
const char\* bracket\_tests[] = {  
 "((()))",  
 "([{}])",  
 "(()",  
 "([)]",  
 "{[()()()]}"  
};  
int num\_bracket\_tests = sizeof(bracket\_tests) / sizeof(bracket\_tests[0]);  
  
for (i = 0; i < num\_bracket\_tests; i++)  
{  
 int is\_balanced = check\_brackets(bracket\_tests[i]);  
 printf(" \"%s\" -> %s\n", bracket\_tests[i],   
 is\_balanced ? "バランス取れています" : "バランスが取れていません");  
}  
printf("\n");  
  
/\* デモ3: 数値の抽出 \*/  
printf("3. 文字列からの数値抽出\n");  
const char\* number\_tests[] = {  
 "価格は1500円、税込み1650円です",  
 "座標: x=10, y=-25, z=100",  
 "電話番号: 03-1234-5678"  
};  
int num\_number\_tests = sizeof(number\_tests) / sizeof(number\_tests[0]);  
  
for (i = 0; i < num\_number\_tests; i++)  
{  
 int numbers[20];  
 int count = extract\_numbers(number\_tests[i], numbers, 20);  
 printf(" \"%s\"\n", number\_tests[i]);  
 printf(" 抽出された数値: ");  
 for (j = 0; j < count; j++)  
 {  
 printf("%d", numbers[j]);  
 if (j < count - 1) printf(", ");  
 }  
 printf("\n\n");  
}  
  
/\* デモ4: ワイルドカードマッチング \*/  
printf("4. ワイルドカードマッチング\n");  
const char\* match\_tests[][2] = {  
 {"hello", "h\*o"},  
 {"test123", "test\*"},  
 {"filename.txt", "\*.txt"},  
 {"abc", "a?c"},  
 {"hello", "world"}  
};  
int num\_match\_tests = sizeof(match\_tests) / sizeof(match\_tests[0]);  
  
for (i = 0; i < num\_match\_tests; i++)  
{  
 int matches = wildcard\_match(match\_tests[i][0], match\_tests[i][1]);  
 printf(" \"%s\" と \"%s\" -> %s\n",   
 match\_tests[i][0], match\_tests[i][1],  
 matches ? "マッチします" : "マッチしません");  
}  
printf("\n");  
  
/\* デモ5: 文字列の置換 \*/  
printf("5. 文字列の置換\n");  
char replace\_test[] = "Hello world! This is a test world.";  
printf(" 元の文字列: \"%s\"\n", replace\_test);  
int replacements = string\_replace(replace\_test, "world", "universe");  
printf(" 置換後: \"%s\"\n", replace\_test);  
printf(" 置換回数: %d回\n\n", replacements);  
  
/\* デモ6: 文字列の中央揃え \*/  
printf("6. 文字列の中央揃え\n");  
const char\* center\_tests[] = {"Hello", "World", "Test"};  
int widths[] = {10, 15, 20};  
  
for (i = 0; i < 3; i++)  
{  
 char centered[50];  
 center\_string(centered, center\_tests[i], widths[i]);  
 printf(" 幅%d: |%s|\n", widths[i], centered);  
}  
printf("\n");  
  
/\* デモ7: CSV解析 \*/  
printf("7. CSV形式の解析\n");  
const char\* csv\_tests[] = {  
 "名前,年齢,職業",  
 "田中太郎,25,エンジニア",  
 "\"佐藤,花子\",30,\"営業,マネージャー\""  
};  
int num\_csv\_tests = sizeof(csv\_tests) / sizeof(csv\_tests[0]);  
  
for (i = 0; i < num\_csv\_tests; i++)  
{  
 char fields[10][100];  
 int field\_count = parse\_csv\_line(csv\_tests[i], fields, 10);  
 printf(" \"%s\"\n", csv\_tests[i]);  
 printf(" フィールド: ");  
 for (j = 0; j < field\_count; j++)  
 {  
 printf("[%s]", fields[j]);  
 if (j < field\_count - 1) printf(" ");  
 }  
 printf("\n\n");  
}  
  
printf("=== 高度な文字列処理デモ完了 ===\n");  
return 0;

}

/\* 学習ポイント: 1. 文字列の動的な解析と分割 2. 構文解析の基本（括弧のバランス） 3. パターンマッチングアルゴリズム 4. 文字列操作の効率的な実装 5. フォーマット処理とパース処理

実装のポイント: - C90準拠のため、変数は関数先頭で宣言 - 安全な文字列操作（バッファオーバーフロー対策） - 効率的なアルゴリズムの選択 - エラーハンドリングの考慮 \*/```

## ex9\_9\_sorting\_algorithms.c

```c / ファイル名: ex9\_9\_sorting\_algorithms.c \* 演習9-9: ソートアルゴリズム関数 \* 説明: 各種ソートアルゴリズムの実装（バブル、選択、挿入、マージソート等） \* 規格: C90準拠 \*/ #include <stdio.h>

/\* 関数プロトタイプ \*/ void bubble\_sort(int arr[], int size); void selection\_sort(int arr[], int size); void insertion\_sort(int arr[], int size); void merge\_sort(int arr[], int left, int right); void merge(int arr[], int left, int mid, int right); void quick\_sort(int arr[], int left, int right); int partition(int arr[], int left, int right); void sort\_strings(char strings[][100], int count); void print\_array(int arr[], int size); void copy\_array(int dest[], int src[], int size);

/\* バブルソート \*/ void bubble\_sort(int arr[], int size) { int i, j, temp; int swapped;

for (i = 0; i < size - 1; i++)  
{  
 swapped = 0;  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 /\* 要素を交換 \*/  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 swapped = 1;  
 }  
 }  
 /\* 交換が発生しなかった場合、ソート完了 \*/  
 if (!swapped) break;  
}

}

/\* 選択ソート \*/ void selection\_sort(int arr[], int size) { int i, j, min\_idx, temp;

for (i = 0; i < size - 1; i++)  
{  
 min\_idx = i;  
 for (j = i + 1; j < size; j++)  
 {  
 if (arr[j] < arr[min\_idx])  
 {  
 min\_idx = j;  
 }  
 }  
   
 /\* 最小値を先頭に移動 \*/  
 if (min\_idx != i)  
 {  
 temp = arr[i];  
 arr[i] = arr[min\_idx];  
 arr[min\_idx] = temp;  
 }  
}

}

/\* 挿入ソート \*/ void insertion\_sort(int arr[], int size) { int i, j, key;

for (i = 1; i < size; i++)  
{  
 key = arr[i];  
 j = i - 1;  
   
 /\* key より大きい要素を後ろに移動 \*/  
 while (j >= 0 && arr[j] > key)  
 {  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 arr[j + 1] = key;  
}

}

/\* マージソート用のマージ関数 */ void merge(int arr[], int left, int mid, int right) { int left\_arr[1000]; /* 左の部分配列 */ int right\_arr[1000]; /* 右の部分配列 \*/ int left\_size = mid - left + 1; int right\_size = right - mid; int i, j, k;

/\* 左の部分配列をコピー \*/  
for (i = 0; i < left\_size; i++)  
{  
 left\_arr[i] = arr[left + i];  
}  
  
/\* 右の部分配列をコピー \*/  
for (j = 0; j < right\_size; j++)  
{  
 right\_arr[j] = arr[mid + 1 + j];  
}  
  
/\* マージ処理 \*/  
i = 0; j = 0; k = left;  
while (i < left\_size && j < right\_size)  
{  
 if (left\_arr[i] <= right\_arr[j])  
 {  
 arr[k] = left\_arr[i];  
 i++;  
 }  
 else  
 {  
 arr[k] = right\_arr[j];  
 j++;  
 }  
 k++;  
}  
  
/\* 残りの要素をコピー \*/  
while (i < left\_size)  
{  
 arr[k] = left\_arr[i];  
 i++;  
 k++;  
}  
  
while (j < right\_size)  
{  
 arr[k] = right\_arr[j];  
 j++;  
 k++;  
}

}

/\* マージソート \*/ void merge\_sort(int arr[], int left, int right) { int mid;

if (left < right)  
{  
 mid = left + (right - left) / 2;  
   
 /\* 左半分をソート \*/  
 merge\_sort(arr, left, mid);  
   
 /\* 右半分をソート \*/  
 merge\_sort(arr, mid + 1, right);  
   
 /\* マージ \*/  
 merge(arr, left, mid, right);  
}

}

/\* クイックソート用の分割関数 */ int partition(int arr[], int left, int right) { int pivot = arr[right]; /* 最後の要素をピボットとする \*/ int i = left - 1; int j, temp;

for (j = left; j <= right - 1; j++)  
{  
 if (arr[j] < pivot)  
 {  
 i++;  
 temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
}  
  
temp = arr[i + 1];  
arr[i + 1] = arr[right];  
arr[right] = temp;  
  
return i + 1;

}

/\* クイックソート \*/ void quick\_sort(int arr[], int left, int right) { int pivot\_index;

if (left < right)  
{  
 pivot\_index = partition(arr, left, right);  
   
 /\* ピボットの左側をソート \*/  
 quick\_sort(arr, left, pivot\_index - 1);  
   
 /\* ピボットの右側をソート \*/  
 quick\_sort(arr, pivot\_index + 1, right);  
}

}

/\* 文字列配列のソート（バブルソート版） \*/ void sort\_strings(char strings[][100], int count) { int i, j, k; char temp[100]; int compare\_result;

for (i = 0; i < count - 1; i++)  
{  
 for (j = 0; j < count - 1 - i; j++)  
 {  
 /\* 文字列比較 \*/  
 compare\_result = 0;  
 for (k = 0; strings[j][k] != '\0' && strings[j + 1][k] != '\0'; k++)  
 {  
 if (strings[j][k] < strings[j + 1][k])  
 {  
 compare\_result = -1;  
 break;  
 }  
 else if (strings[j][k] > strings[j + 1][k])  
 {  
 compare\_result = 1;  
 break;  
 }  
 }  
   
 if (compare\_result == 0)  
 {  
 if (strings[j][k] != '\0') compare\_result = 1;  
 else if (strings[j + 1][k] != '\0') compare\_result = -1;  
 }  
   
 if (compare\_result > 0)  
 {  
 /\* 文字列を交換 \*/  
 for (k = 0; strings[j][k] != '\0'; k++)  
 {  
 temp[k] = strings[j][k];  
 }  
 temp[k] = '\0';  
   
 for (k = 0; strings[j + 1][k] != '\0'; k++)  
 {  
 strings[j][k] = strings[j + 1][k];  
 }  
 strings[j][k] = '\0';  
   
 for (k = 0; temp[k] != '\0'; k++)  
 {  
 strings[j + 1][k] = temp[k];  
 }  
 strings[j + 1][k] = '\0';  
 }  
 }  
}

}

/\* 配列を表示する関数 \*/ void print\_array(int arr[], int size) { int i; printf(“[”); for (i = 0; i < size; i++) { printf(”%d”, arr[i]); if (i < size - 1) printf(”, ”); } printf(”]”); }

/\* 配列をコピーする関数 \*/ void copy\_array(int dest[], int src[], int size) { int i; for (i = 0; i < size; i++) { dest[i] = src[i]; } }

int main(void) { printf(“=== ソートアルゴリズムのデモ ===”);

/\* テスト用データ \*/  
int original\_data[] = {64, 34, 25, 12, 22, 11, 90, 88, 76, 50};  
int data\_size = sizeof(original\_data) / sizeof(original\_data[0]);  
int test\_data[20];  
int i;  
  
printf("元のデータ: ");  
print\_array(original\_data, data\_size);  
printf("\n\n");  
  
/\* 1. バブルソート \*/  
printf("1. バブルソート\n");  
copy\_array(test\_data, original\_data, data\_size);  
printf(" ソート前: ");  
print\_array(test\_data, data\_size);  
printf("\n");  
bubble\_sort(test\_data, data\_size);  
printf(" ソート後: ");  
print\_array(test\_data, data\_size);  
printf("\n\n");  
  
/\* 2. 選択ソート \*/  
printf("2. 選択ソート\n");  
copy\_array(test\_data, original\_data, data\_size);  
printf(" ソート前: ");  
print\_array(test\_data, data\_size);  
printf("\n");  
selection\_sort(test\_data, data\_size);  
printf(" ソート後: ");  
print\_array(test\_data, data\_size);  
printf("\n\n");  
  
/\* 3. 挿入ソート \*/  
printf("3. 挿入ソート\n");  
copy\_array(test\_data, original\_data, data\_size);  
printf(" ソート前: ");  
print\_array(test\_data, data\_size);  
printf("\n");  
insertion\_sort(test\_data, data\_size);  
printf(" ソート後: ");  
print\_array(test\_data, data\_size);  
printf("\n\n");  
  
/\* 4. マージソート \*/  
printf("4. マージソート\n");  
copy\_array(test\_data, original\_data, data\_size);  
printf(" ソート前: ");  
print\_array(test\_data, data\_size);  
printf("\n");  
merge\_sort(test\_data, 0, data\_size - 1);  
printf(" ソート後: ");  
print\_array(test\_data, data\_size);  
printf("\n\n");  
  
/\* 5. クイックソート \*/  
printf("5. クイックソート\n");  
copy\_array(test\_data, original\_data, data\_size);  
printf(" ソート前: ");  
print\_array(test\_data, data\_size);  
printf("\n");  
quick\_sort(test\_data, 0, data\_size - 1);  
printf(" ソート後: ");  
print\_array(test\_data, data\_size);  
printf("\n\n");  
  
/\* 6. 文字列ソート \*/  
printf("6. 文字列ソート\n");  
char fruits[][100] = {"orange", "apple", "banana", "grape", "cherry"};  
int fruit\_count = 5;  
  
printf(" ソート前: ");  
for (i = 0; i < fruit\_count; i++)  
{  
 printf("%s", fruits[i]);  
 if (i < fruit\_count - 1) printf(", ");  
}  
printf("\n");  
  
sort\_strings(fruits, fruit\_count);  
  
printf(" ソート後: ");  
for (i = 0; i < fruit\_count; i++)  
{  
 printf("%s", fruits[i]);  
 if (i < fruit\_count - 1) printf(", ");  
}  
printf("\n\n");  
  
/\* 7. 性能比較用のより大きなデータセット \*/  
printf("7. 大きなデータセットでのテスト\n");  
int large\_data[50];  
int large\_size = 50;  
  
/\* テストデータを生成（擬似ランダム） \*/  
printf(" 生成されたデータ（最初の20個）: ");  
for (i = 0; i < large\_size; i++)  
{  
 large\_data[i] = (i \* 17 + 23) % 100; /\* 簡単な擬似ランダム \*/  
 if (i < 20)  
 {  
 printf("%d", large\_data[i]);  
 if (i < 19) printf(", ");  
 }  
}  
printf("...\n");  
  
/\* 挿入ソートでソート \*/  
insertion\_sort(large\_data, large\_size);  
  
printf(" ソート後（最初の20個）: ");  
for (i = 0; i < 20; i++)  
{  
 printf("%d", large\_data[i]);  
 if (i < 19) printf(", ");  
}  
printf("...\n\n");  
  
printf("=== ソートアルゴリズムデモ完了 ===\n");  
return 0;

}

/\* 学習ポイント: 1. 基本的なソートアルゴリズムの理解と実装 2. 各アルゴリズムの時間計算量の違い 3. 分割統治法（マージソート、クイックソート） 4. 文字列ソートの実装方法 5. アルゴリズムの選択基準

実装のポイント: - C90準拠のため、変数は関数先頭で宣言 - 各ソートの特徴を活かした実装 - 効率性と可読性のバランス - メモリ使用量の考慮 \*/```